Semestrální úloha č. 18
Fast Insertion-Based Optimization of Bounding Volume Hierarchies

Martin Čáp*
Katedra počítačové grafiky a interakce,
Fakulta elektrotechnická, ČVUT Praha

13. května 2018

Abstrakt

Cílem této práce je implementace optimalizačního algoritmu pro hierarchie obálek (bounding volume hierarchies - BVHs). Algoritmus byl aplikován na hierarchie obálek, jež byly postaveny dělením pomocí prostorového mediánu (oproti použití cenového modelu zvaného surface area heuristic - SAH). Optimalizovaná struktura byla otestována na scénách různé komplexity a byly naměřeny důležité údaje jako je cena před a po optimalizaci, doba vykreslování s původním a optimalizovaným BVH, počet traverzačních kroků anebo např. počet incidenčních operací.

klíčová slova: hierarchie obálek, BVH, optimalizace, surface area heuristic, SAH, sledování paprsku, raytracing

1 Úvod

Rychlost metody sledování paprsků (raytracing) je závislá na kvalitních akceleračních strukturách, které nám umožňují vykreslit komplexní scény v relativně krátkých časech. Mým úkolem bylo implementovat algoritmus, jenž vylepšuje již existující datovou strukturu, konkrétně hierarchii obálek. Hierarchie obálek je před optimalizací postavena pomocí dělení prostoru podle jeho mediánu. Volba osy pro dělení je založena na metodě *round robin*, tzn., že se dělící osy pravidelně střídají v každé úrovni hierarchie (x, y, z, x, y, z, ...). První dělící osa je zvolena podle rozsahu obálky celé scény (metoda *max extent*).

Samotný algoritmus je postaven na cenovém modelu SAH (surface area heuristic), který aproximuje průměrný počet operací (traverzačních kroků a incidenčních operací) na jeden paprsek. Metoda je vhodná především na architektonické scény s vysokou komplexitou, především v případech, kdy je předpokládaná doba vykreslování scény vysoká (např. řádově v hodinách). Algoritmus provádí globální úpravy hierarchie, díky čemuž konverguje velmi rychle k možnému minimu ceny hierarchie oproti algoritmu pana Kenslera z roku 2008 nazvanému simulované žíhání (simulated annealing), jenž konverguje ke stejné nebo horší ceně řádově pomaleji [1, 2].

2 Algoritmus

Jak již bylo řečeno, algoritmus je založen na globálních úpravách hierarchie obálek, díky kterým je celková cena hierarchie postupně snižována. Algoritmus navíc není závislý na zvoleném způsobu stavby hierarchie (např. SAH, prostorový medián, objektový medián a další). Jediný předpoklad pro nás bude, že BVH na vstupu má v každém listu pouze jeden trojúhelník.

*B4M39DPG – Martin Čáp, letní semestr 2017/18

Základní schéma algoritmu vypadá následovně:

Dokud nejsou splněna ukončovací kritéria:

1. Zvol vnitřní uzly pro optimalizaci.
2. Pro každý vnitřní uzel:
   (a) Odeber tento uzel, oba jeho potomky a jeho rodiče ze stromu.
   (b) Najdi vhodnou pozici pro znovuvložení potomků za použití metody větví a mezí založené na cenách.
   (c) Vlož každého z potomků na jeho novou pozici a oprav obálky všech afektovaných uzlů.

2.1 Cenový model SAH

Pro optimalizaci hierarchie vycházíme z cenového modelu surface area heuristic (SAH). Ten určuje průměrný předpokládaný počet operací pro zpracování paprsku v každém uzlu. Vychází ze dvou předpokladů, které většinou v praxi nemusí nutně platit. Předpoklady jsou, že paprsky mají uniformní distribuci a že navíc nejsou zastíněny. Jak lze ale vidět na obrázku č. 6, cenový model vypovídá, i přes nesplnění těchto předpokladů, o výsledné rychlosti struktury, neboť vztah mezi cenou a rychlostí vykreslení je skoro lineární.

Pokud je N námi zkoumaný uzel, jeho cena je pak dána jako

$$C(N) = \begin{cases} c_T + \frac{SA(L(N))\cdot C(L(N)) + SA(R(N))\cdot C(R(N))}{SA(N)} & N \text{ je vnitřní} \\ c_I \cdot t_N & N \text{ je list} \end{cases},$$

kde c_T je cena traverzace včetně výpočtu průniku s obálkou, c_I je cena výpočtu průniku s trojúhelníkem, t_N je počet trojúhelníků v listu N, $SA(x)$ je povrch obálky uzlu x, a $L(N)$ a $R(N)$ jsou levý a pravý potomek uzlu N.

Po odstranění rekurze získáme pro kořen hierarchie T

$$C(T) = \frac{1}{SA(T)} \left[c_T \cdot \sum_{N \in inner\ nodes} SA(N) + c_I \cdot \sum_{N \in leaves} SA(N) \cdot t_N \right],$$

kde $SA(T)$ je povrch obálky celé scény.

Důležité je si všimnout, že pokud je počet trojúhelníků v listech konstatní (což pro nás platí, neboť máme v každém listu jen jeden trojúhelník), pak je naším cílem minimalizovat součet povrchů vnitřních uzlů. Co se týče implementace, je nutné zmínit, že součet povrchů obálek listů hierarchie stačí vypočítat jen jednou před optimalizací (a jednou po procesu slučování uzlů). Součet povrchů vnitřních uzlů je v každém kroku změněn inkrementálním přičítáním/odečítáním rozdílů povrchů nových a starých obálek během jejich přepočtu po provedení změny v hierarchii (vyjmutí nebo vložení uzlů).

2.2 Aktualizace uzlů

Proces aktualizace uzlů se dělí na 3 etapy:

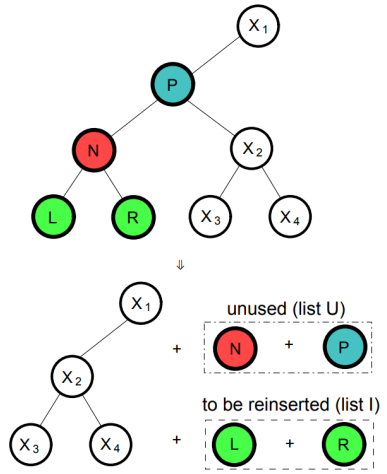
1. Odstranění uzlů
2. Vyhledání nových pozic pro vyjmuté uzly
3. Zpětné vložení do hierarchie

Předpokládejme, že jsme pro následující kapitoly zvolili uzel N. Označme levého a pravého potomka N jako L a R. Označme rodičovský uzel N jako P.

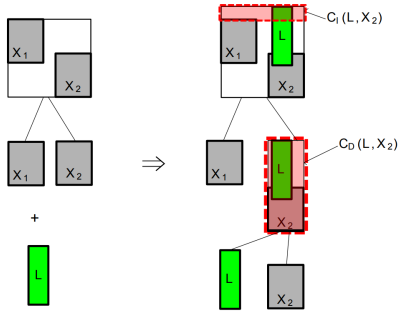
2.2.1 Odstranění uzlů

Odstranění uzlů je přímočaré. Při aktualizaci N odstraníme jak N, tak L, R i P a aktualizujeme strom, aby nebyla porušena jeho topologická struktura. Kromě opravy referencí musíme navíc aktualizovat povrch obálek všech afektovaných uzlů a to traverzací až ke kořeni. Vložíme L a R do uspořádaného seznamu, který obsahuje uzly pro zpětné vložení do hierarchie, vyšší prioritu bude mít uzel s větším povrchem.

Uzly N a P uložíme do odlišného seznamu - budou později sloužit k vložení L a R do jejich nových pozic. Je nutné zmínit, že L a R nemusí být listy, ale celé podstromy. Ve speciálním případě, kdy je aktualizovaný uzel N potomkem kořene, vyjmeme opět N, L, R a kořen samotný s tím, že novým kořenem se stane druhý potomek původního kořene (tedy levý, pokud je N pravým potomkem kořene a naopak). Aktualizace kořene samotného je zakázána.



(a) Lokální situace před a po odstranění uzlu N.



(b) 2D ukázka přímé ceny C_D a indukované ceny C_I při vkládání uzlu L na pozici uzlu X_2.

Obrázek 1: Odstranění uzlů a ceny používáné při vyhledávání nových pozic [1].

2.2.2 Vyhledávání nových pozic

Nechť pro následující kapitoly vkládáme uzel L zpět do hierarchie (tzn. že obálka uzlu L má větší povrch než obálka uzlu R). Pro vyhledání nové pozice pro vyjmutý uzel je použita metoda větví a mezí založená na tzv. přímé a indukované ceně. Hledání je *greedy* (chamtivé), tzn. že zvolíme pozici, která co nejvíce minimalizuje cenu výsledného BVH. Hledání začíná v kořeni stromu a inkrementálně počítá zvětšení povrchu ve zkoumaných uzlech X (v implementaci nazvaných jako kandidáti - candidates), pokud bychom do nich vložili právě aktualizovaný uzel L.

Přímá cena (direct cost) je definována jako

$$C_D(L, X) = SA(X \cup L),$$

kde $SA(X \cup L)$ je velikost povrchu obálky sjednocení obálek uzlů L a X.

Indukovaná cena (induced cost) je akumulované zvýšení povrchu od kořene do rodičovského uzlu právě zkoumaného uzlu X za předpokladu, že uzel L byl vložen do podstromu, jenž má kořen v X. Indukovanou cenu lze popsat rekurzivním vzorcem

$$C_I(L, X) = \begin{cases} 0 & \text{pokud } X \text{ je kořen} \\ C_I(L, p(X)) + SA(p(X) \cup L) - SA(p(X)) & \text{jinak} \end{cases},$$

kde $p(X)$ je rodič uzlu X.

Celkové zvýšení ceny je pak součtem těchto dvou hodnot

$$C(L,X) = C_D(L,X) + C_I(L,X).$$

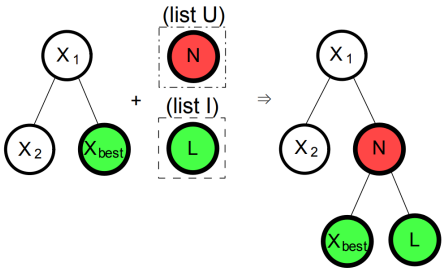
Jak již bylo řečeno, abychom našli uzel X_{best} takový, který minimalizuje tuto hodnotu pro celý strom, použijeme metodu větví a mezí. Metoda větví a mezí je založena na prioritní frontě, kde priorita je inverzně proporcionální indukované ceně, tzn., že čím nižší je indukovaná cena uzlu, tím vyšší je jeho priorita. Hledání můžeme ukončit v případě, kdy současně nejmenší cena C_{best}, korespondující s X_{best}, nemůže být překonána. Celý algoritmus může být ukončen v případě, kdy je dolní mez prvního uzlu (jeho indukovaná cena) v prioritní frontě větší než C_{best}.

```
1  Algorithm:FindNodeForReinsertion(node L)
2  // C_best - the smallest total cost increase found so far
   C_best = infinity;
3  // Priority queue contains pairs: (node, induced cost)
   Push (Root of BVH, 0, 1/ε) to priority queue PQ;
4  while PQ is not empty do
5      (X, C_I(L,X)) = Pop node from PQ;
6      if C_I(L,X) + SA(L) >= C_best then
7          // Early termination - not possible
8          break; // to reduce the cost C_best
9      end
10     // Compute the total cost of merging L with X
11     C_D(L,X) = SA(X ∪ L); // Direct cost
12     C(L,X) = C_I(L,X) + C_D(L,X); // Total cost
13     if C(L,X) < C_best then
14         // Merging L and X decreases the best cost
15         C_best = C(L,X);
16         // X_best is the currently best node found
17         X_best = X;
18     end
19     // Calculate the induced cost for children of X
20     C_I = C(L,X) - SA(X);
21     // Check if the cost decrease is possible in subtree
22     if C_I + SA(L) < C_best then
23         if X is not a leaf then
24             // Search in both children
25             Push (left child of X, C_I, 1/(C_I+ε)) to PQ ;
26             Push (right child of X, C_I, 1/(C_I+ε)) to PQ ;
27         end
28     end
29 end
30 return X_best;
```

Obrázek 2: Pseudokód pro vyhledávání nejlepší pozice pro zpětné vložení uzlu L [1].

2.2.3 Zpětné vložení uzlů

Po zvolení uzlu X_{best} pro znovuvložení uzlu L jednoduše spojíme X_{best} a L a použijeme N nebo P jako jejich společného rodiče. Po každém vložení musíme aktualizovat povrchy všech obálek od společného rodiče až ke kořeni. Důležité je též připomenout, že během aktualizace povrchů obálek je nutné aktualizovat proměnnou, obsahující součet povrchů obálek všech vnitřních uzlů, o rozdíl nové a staré hodnoty.



Obrázek 3: Ilustrace situace, kdy vkládáme zpět uzel L na pozici X_{best} do stromu hierarchie. Zde za použití N jako společného rodiče [1].

2.3 Volba uzlů k aktualizaci

Volba uzlů k aktualizaci je velmi důležitou součástí tohoto algoritmu, na které závisí, jak rychle bude cena hierarchie konvergovat ke globálnímu minimu. První možnost, která se nabízí, je náhodný výběr (random sampling). Ten sice logicky konverguje ke globálnímu minimu, ale velmi pomalu. V algoritmu jsou použity tzv. hodnoty neefektivity uzlů (inefficiency measures), které korelují s možným snížením celkové ceny hierarchie.

První z nich je M_{SUM},

$$M_{SUM}(N) = \frac{SA(N)}{\frac{1}{|children\ of\ N|} \cdot \sum_{X \in children\ of\ N} SA(X)}.$$

Ta odhalí případy, kdy uzel N má velkou obálku, zatímco jeho potomci je mají velmi malé a díky tomu je v N mnoho prázdného místa. V takovém případě je hodnota M_{SUM} vysoká.

Druhá je M_{MIN},

$$M_{MIN}(N) = \frac{SA(N)}{min_{X \in children\ of\ N} SA(X)}.$$

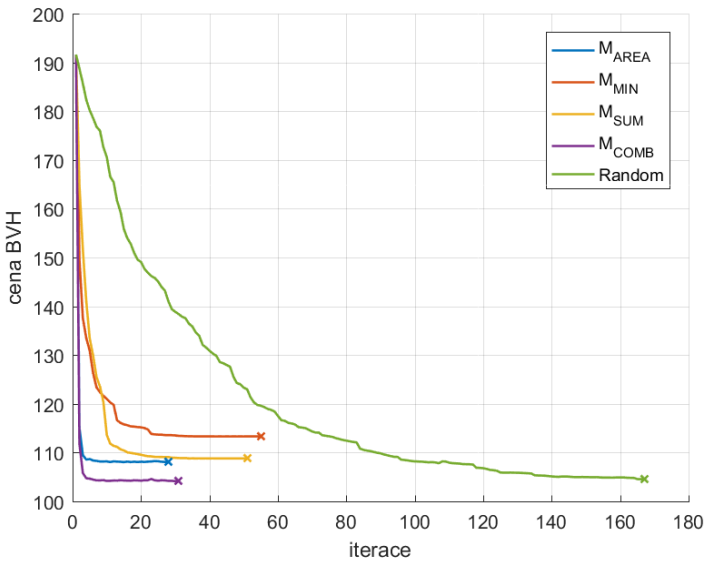
M_{MIN} rozpozná případy, kdy mají potomci uzlu N diametrálně odlišný povrch obálky. Např. jeden potomek je terén celé scény, zatímco druhý je automobil, který je 100× menší.

Třetí, M_{AREA}, dává přednost uzlům s vysokým povrchem a platí

$$M_{AREA}(N) = SA(N).$$

Společně jsou tyto hodnoty neefektivity schopny detekovat mnoho situací, ve kterých může dojít, po volbě a vylepšení uzlu, ke značnému zlepšení globální ceny hierarchie. K nejrychlejšímu snížení celkové ceny hierarchie dochází za použití součinu těchto hodnot zvaného M_{COMB},

$$M_{COMB}(N) = M_{SUM}(N) \cdot M_{MIN}(N) \cdot M_{AREA}(N).$$



Obrázek 4: Porovnání strategií volby uzlů pro aktualizaci. Pro všechny byla použita ukončovací hodnota $p_T = 10$. Můžeme si všimnout, že M_{COMB} a náhodná volba uzlů (*Random*) konvergují k nejmenší hodnotě, kde se M_{COMB} hned po prvních krocích dostává do globálního minima. Měřeno na scéně konferenční místnosti (conference).

2.4 Aktualizace uzlů - hlavní cyklus

Nyní víme, jak aktualizovat jeden uzel a jak ho vybrat. Na závěr se podíváme na to, jak by měla aktualizace uzlů probíhat. Jak bylo řečeno v úvodu, práce algoritmu je rozdělena do iterací. V každé iteraci vybereme $k\%$ vnitřních uzlů, které aktualizujeme (provádíme tzv. batch processing). V hlavním cyklu tedy spočítáme pro všechny vnitřní uzly jejich hodnotu neefektivity M_{COMB}, přičemž si zapamatujeme $k\%$ z nich do seznamu (candidates v kódu). Tento seznam seřadíme tak, abychom postupně vybírali uzly s nejvyšší hodnotou M_{COMB}. Aktualizujeme všechny vybrané uzly pomocí již popsaných kroků. Po každé iteraci porovnáme současnou cenu hierarchie s tou předchozí a počítáme počet iterací, které cenu nezlepšily (zůstala stejná, nebo se zvýšila). Protože by se mohlo stát, že při výběru uzlů podle M_{COMB} uvázneme v lokálním minimu (některé uzly nikdy nemusíme vybrat), přepneme způsob výběru uzlů po p_R iteracích, jež nezlepšily cenu, na náhodný výběr uzlů. Je důležité čtenáře upozornit, že počítadlo nezlepšujících se iterací se nevynuluje. Jakmile počítadlo dosáhne hodnoty p_T, je celý optimalizační algoritmus ukončen. Samozřejmě musí platit $p_T > p_R$. Pokud by uživatel nastavil $p_R = 0$, pak by byl náhodný výběr použit hned od začátku. Pokud by naopak platilo $p_R \geq p_T$, pak by náhodný výběr nebyl použit nikdy.

2.5 Slučování uzlů

Jako poslední optimalizační krok se nabízí slučování více listů a vnitřních uzlů v jeden list, který by obsahoval více jak jeden trojúhelník. Tento krok lze provést v lineárním čase, vzhledem k počtu uzlů hierarchie. Pro sloučení uzlů je potřeba projít strom v post order pořadí, v každém vnitřním uzlu spočítat počet trojúhelníků, jež mu náleží, a porovnat jeho cenu s potenciálním listem, který by vznikl sloučením zkoumaného uzlu a jeho podstromů. Pokud je cena tohoto potenciálního listu menší než cena vnitřního uzlu, provedeme sloučení v jeden list a uložíme v něm odkaz na všechny trojúhelníky, jež do něj patří.

2.6 Vlastní nástroje

Kromě hlavního algoritmu byla vytvořena i vlastní aplikace RT05, jež slouží k testování a měření atributů hierarchie. Nastavení pro tuto aplikaci lze najít v souboru prostředí (environment file) pod pojmem Application. V hierarchii je navíc mnoho definovaných maker, jež umožňují uživateli (který má možnost aplikaci zkompilovat) změnit další nastavení hierarchie podmíněným překladem. Byla vytvořena alternativní implementace optimalizačního procesu (pomalejší), která nám ale umožňuje provést měření rychlosti vykreslování pro hierarchii po libovolném množství optimalizačních kroků/iterací. Díky tomu můžeme testovat jednotlivé strategie volby hodnoty neefektivity (ineffectivity measure) uzlů pro aktualizaci.

3 Potíže při implementaci

Myslím, že implementace probíhala nad očekávání hladce, nicméně, problémy se rovněž vyskytly. Jedním z nich byla nechtěnná volba kořene pro optimalizaci. K této volbě totiž nedocházelo "při volbě" kandidátů na optimalizaci, ale během postupného vkládání již vybraných kandidátů, kdy došlo k přestavění stromu tak, že další kandidát byl kořenem upraveného stromu z předchozího kroku. K tomuto problému ve zkratce docházelo z toho důvodu, že jsem si neuvědomil, jaké důsledky má zpracování více uzlů v jedné iteraci (batch processing).

Samozřejmě největším problémem v tomto semestru byl čas, což hrálo i velkou roli při implementaci. Na jednu stranu musím říci, že nedostatek času mě donutil vyhnout se zbytečné předčasné optimalizaci ("Premature optimization is the root of all evil." - Donald Knuth). Naopak bych se ale rád práci věnoval více, neboť je stále co zlepšovat.

Jednou z dalších chyb, kterých jsem se na začátku dopustil, byl výpočet sumy povrchu obálek vnitřních uzlů v každé iteraci průchodem celého stromu. To je extrémně neefektivní a jedná se o hrubou chybu při implementaci tohoto algoritmu. Rychle mi došlo, že při přepočítávání obalových těles bude rozdíl povrchů nové a původní obálky přičten (respektive odečten, pokud je nová obálka menší) od současné sumy povrchů obálek vnitřních uzlů. To je v implementaci vidět ve funkci Recalculate ve struktuře SBBox, jež má jako návratovou hodnotu tento rozdíl.

Posledním problémem, pokud se tedy opravdu jedná o problém, bylo slučování vnitřních uzlů hierarchie pro snížení ceny v posledním (post process) kroku optimalizace. Problémem nebyl samotný

algoritmus na slučování, jenž je založený na post order průchodu stromu a je relativně jednoduchý na pochopení. Byly to naměřené časy pro hierarchii, která tento krok použila. Samotná cena hierarchie se opravdu podle očekávání snížila řádově o dalších 10%, ale časy vykreslování scén se buď zhoršily, nebo zůstaly stejné. Podle mých měření totiž, díky slučovacímu kroku, sice uděláme méně traverzačních kroků, ale za cenu výrazně více incidenčních operací v listech (neboť pokud se jedná o list, musíme vyzkoušet všechny trojúhelníky v něm). Pokusil jsem se redukovat volbu uzlů pro sloučení konstantou cCompact, jež zvýšila potenciální ceny sloučených uzlů. Navíc jsem omezil maximální počet trojúhelníků, jež mohou být v listech. Bohužel ani jedna z těchto úprav neměla výrazný dopad na naměřené časy, které byly jen ve výjimečných případech lepší než pro hierarchii s jedním trojúhelníkem na list.

3.1 Časové nároky

Na semestrální prací jsem strávil cca 70 hodin, včetně psaní tohoto dokumentu a tvorby prezentací. Na první prezentaci jsem strávil mnoho času, neboť jsem se snažil pochopit veškerou teorii popsanou v článku před zahájením implementace. Překvapivě mnoho času mi též zabrala příprava rozhraní pro testování aplikace, které umožňuje uživateli v environment souboru nastavit pro jakou hierarchii (originální, optimalizovanou, se sloučenými uzly) chce měřit s tím, že všechny údaje budou uloženy do příslušných logovacích souborů v přehledné formě.

4 Naměřené výsledky

Závěrečné měření bylo provedeno ve školní učebně KN:E-327 na počítači s procesorem Intel Xeon E3-1231 v3 3.40GHz, hlavní pamětí 16GB, grafickou kartou GeForce GTX 750 Ti 2GB GDDR5 a operačním systémem Windows 7. Aplikace byla kompilována v Microsoft Visual Studiu 2015, v Release nastavení (Maximize Speed - /O2) pro 64 bitové systémy.

Pro měření byly použity hodnoty proměnných $k = 1\%$, $p_R = 5$ a $p_T = 10$ s výběrem podle M_{COMB}. Navíc byly nastaveny hodnoty $c_{compact} = 1.25$ a $maxTrisInLeaf = 8$. Scény byly měřeny s primárními paprsky.

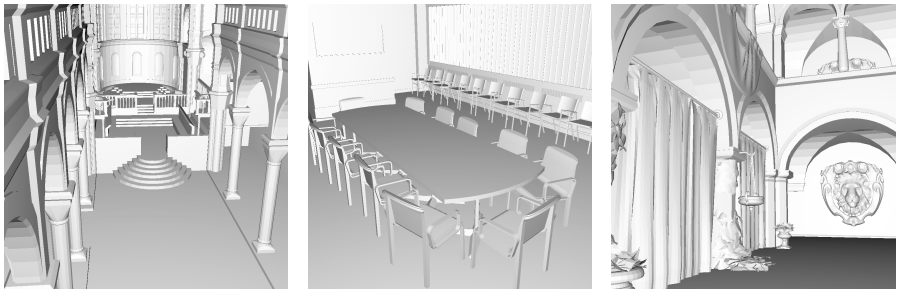
Byla použita jedna scéna navíc, která nebyla v dodaných scénách na stránkách předmětu. Jedná se o scénu paláce Sponza v Dobrovniku od firmy Crytek, kterou lze získat na adrese `http://www.crytek.com/cryengine/cryengine3/downloads`.

4.1 Diskuze

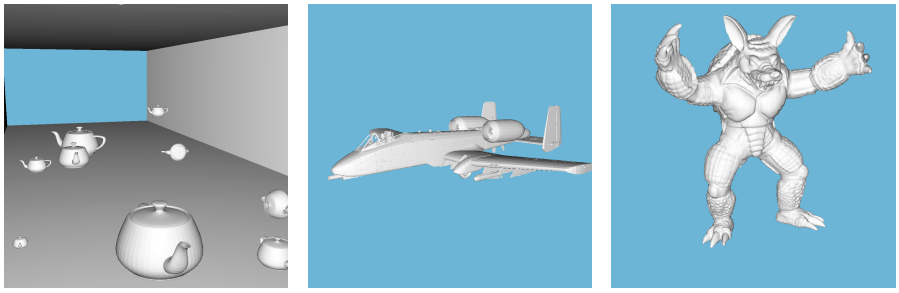
Jak lze vidět v tabulce, výsledné ceny všech scén po optimalizaci byly výrazně sníženy, v některých případech až na polovinu ceny původní. Zrychlení můžeme pozorovat především v architektonických scénách, pro které je tato optimalizační metoda vhodná. U všech scén můžeme pozorovat snížení průměrného počtu traverzačních operací, až na scény s jedním složitým objektem.

Při porovnání svých výsledků s výsledky uvedenými v článku jedna věc neodpovídá, a to cena naivně postaveného BVH dělením podle prostorového mediánu, jež v mé implementaci vychází mnohem lepší (např. 109.0 v mé implementaci a 185.0 ve článku pro scénu Forest Fairy). Naopak můžeme pozorovat, že cena optimalizovaného BVH odpovídá cenám uvedeným v článku. Např. 70.5 v mé implementaci a 71.3 v článku pro scénu katedrály v Šibeniku, 103.0 a 102.7 pro scénu konferenční místnosti (conference), nebo 92.0 a 92.9 pro již zmiňovanou scénu Forest Fairy.

Bohužel u scén, jež obsahují jeden složitý objekt, jsem naopak zaznamenal zpomalení vykreslování, přestože cena BVH byla snížena průměrně o 10%. Tento jev můžeme pozorovat u dvou scén: Armadillo a Asian Dragon.

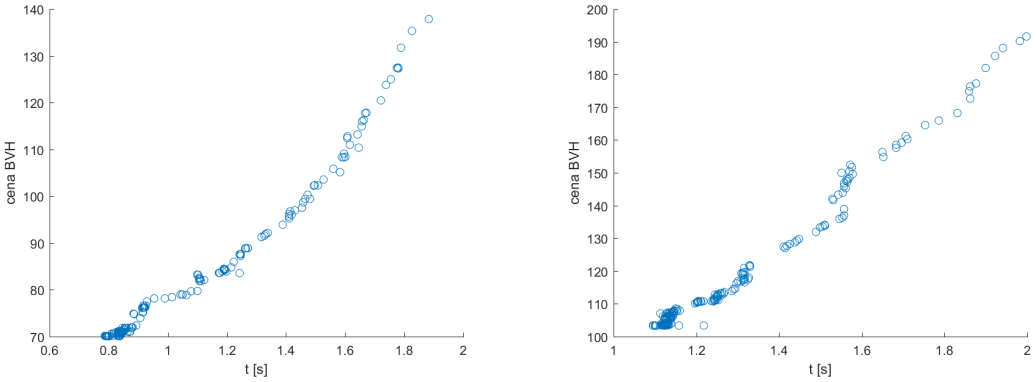


(a) Šibenik (b) Conference (c) Sponza



(d) Teapots (e) A-10 Warthog (f) Armadillo

Obrázek 5: Vybrané obrázky vytvořené pomocí optimalizovaného BVH.



(a) Teapots (b) Conference

Obrázek 6: Měření korelace ceny BVH a času vykreslení obrázku u dvou scén. Můžeme vidět téměř lineární závislost těchto dvou atributů.

5 Závěr

Byl implementován algoritmus na optimalizaci hierarchií obálek, jenž vychází z cenového modelu SAH. Algoritmus iterativně redukuje cenu výsledného BVH a tím snižuje průměrný počet traverzačních kroků a incidenčních operací. Díky tomu můžeme u architektonických scén pozorovat výrazné zrychlení vykreslování, např. z původních 1.876 [s] na 0.715 [s] pro scénu čajových konvic (teapots), nebo z 2.532 [s] na 1.558 [s] pro scénu katedrály v Šibeniku. Redukce ceny odpovídá hodnotám uvedených v referenčním článku [1]. Rychlost optimalizace též přibližně odpovídá rychlosti uvedené ve článku, vezmeme-li do úvahy jiné výchozí BVH a rychlost uvedených strojů, na kterých bylo měření provedeno.

Reference

[1] Bittner, J., Hapala, M., a Havran, V.: Fast Insertion-Based Optimization of Bounding Volume Hierarchies. *Computer Graphics Forum 32* (2013), 1, pp. 85-100

[2] Kensler A.: Tree Rotations for Improving Bounding Volume Hierarchies. *2008 IEEE Symposium on Interactive Ray Tracing* (2008), pp. 73-76

M	cena BVH	doba stavby [s]	doba optimalizace [s]	doba vykreslení [s]	prům. # trav. kroků	prům. # inc. operací
A-10 Warthog (219 tis. troj.)						
1	92.34	0.09	–	0.356	15.56	1.01
2	55.36	5.74	5.65	0.329	12.38	1.15
3	49.89	5.80	0.05	0.326	10.81	2.81
Armadillo (345 tis. troj.)						
1	102.14	0.18	–	0.582	24.07	1.27
2	88.04	7.49	7.31	0.745	27.02	1.49
3	84.61	7.56	0.07	0.764	25.37	3.53
City 1 (68 tis. troj.)						
1	114.30	0.03	–	0.582	31.87	3.08
2	74.05	1.22	1.19	0.461	21.75	2.70
3	63.64	1.24	0.02	0.459	18.37	6.22
City 2 (75 tis. troj.)						
1	141.67	0.03	–	0.794	41.80	3.31
2	67.33	1.58	1.55	0.624	29.85	2.64
3	60.28	1.61	0.03	0.658	25.65	9.51
Conference (283 tis. troj.)						
1	191.61	0.12	–	2.012	129.13	12.37
2	102.99	4.88	4.76	1.136	62.12	9.88
3	84.87	4.94	0.07	1.116	49.15	21.51
Forest Fairy (174 tis. troj.)						
1	109.07	0.08	–	1.171	73.75	5.43
2	92.04	1.29	1.21	1.018	56.65	5.90
3	76.76	1.33	0.04	1.020	47.50	14.61
Park (29 tis. troj.)						
1	79.41	0.01	–	1.039	73.73	5.78
2	54.25	0.23	0.22	0.872	58.60	6.40
3	48.84	0.24	0.01	0.869	50.15	12.73
Šibenik (80 tis. troj.)						
1	125.89	0.03	–	2.532	175.37	9.97
2	70.49	1.95	1.92	1.558	98.80	6.59
3	62.24	1.98	0.03	1.574	92.31	13.92
Sponza (279 tis. troj.)						
1	432.25	0.12	–	4.326	293.77	24.49
2	196.74	8.24	8.12	1.689	99.22	8.76
3	157.74	8.31	0.06	1.728	87.33	27.11
Teapots (201 tis. troj.)						
1	137.84	0.08	–	1.876	134.05	15.92
2	70.16	3.61	3.53	0.715	40.23	3.76
3	54.21	3.66	0.05	0.812	34.90	19.45
Asian Dragon (7219 tis. troj., $k = 0.4$, $p_R = 1$, $p_T = 2$)						
1	108.42	2.88	–	0.857	29.64	1.37
2	97.28	137.99	135.11	1.154	34.16	1.87
3	96.66	139.12	1.13	1.304	32.66	3.65

Tabulka 1: Výsledky z měření. Metoda 1 (M = 1) je BVH postavené pomocí dělení v prostorovém mediánu s round robin volbou dělící osy (až na kořen, kde je osa zvolena podle obalového tělesa). M = 2 jsou výsledky pro optimalizované BVH pomocí popsané metody bez sloučených uzlů (právě jeden trojúhelník v každém listu) a M = 3 se sloučenými uzly. Doba stavby je celková doba včetně postavení původního BVH.