Fast Insertion-Based Optimization of Bounding Volume Hierarchies

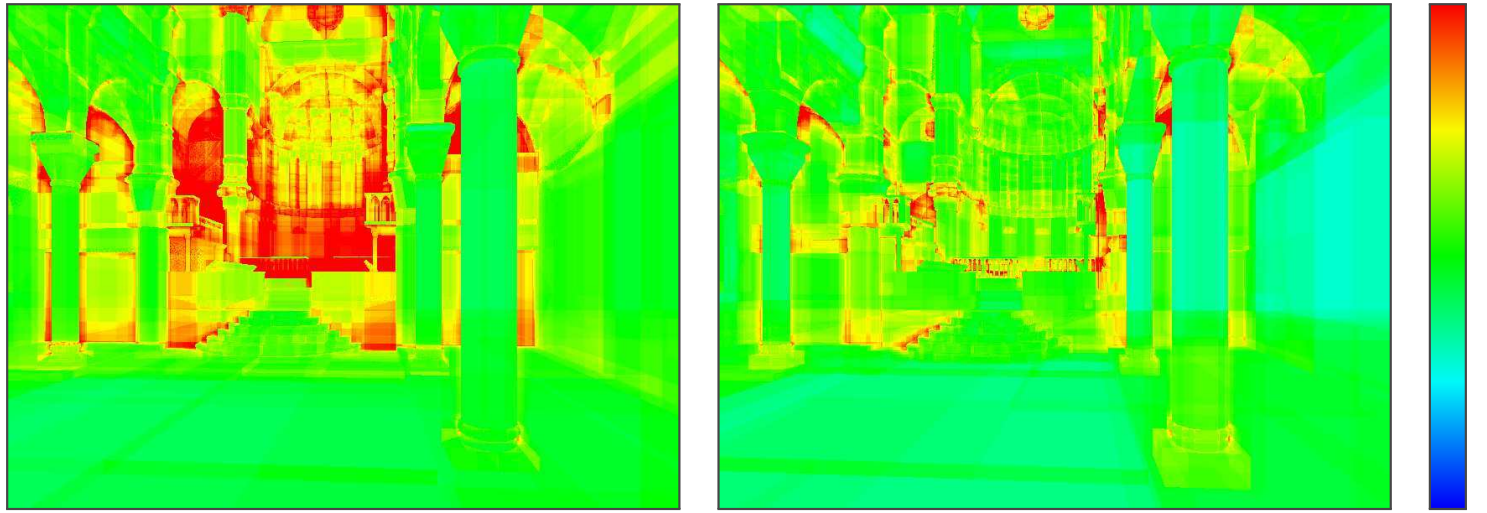
Martin Čáp

Obsah

- Motivace
- Algoritmus
- Výsledky

Motivace

- Raytracing závislý na kvalitních akceleračních strukturách
- Metoda pro vylepšení hierarchií obálek (Bounding Volume Hierarchy - BVH)
- Vhodné na architektonické scény
  - Zrychlení oproti simulovanému žíhání - Kensler 2008 [2]
  - Podle autorů 25-147x rychlejší
  - Dosažení BVH podobné kvality jako u sim. žíhání



(a) (b) (c)

Obrázek 1: Vizualizace počtu traverzačních kroků s primárními a stínovými paprsky Katedrály sv. Jakuba (Šibenik) [1].
(a) BVH postavené s SAH
(b) BVH optimalizované touto metodou
(c) Spektrum: modrá je 0 kroků, červená 100 a více

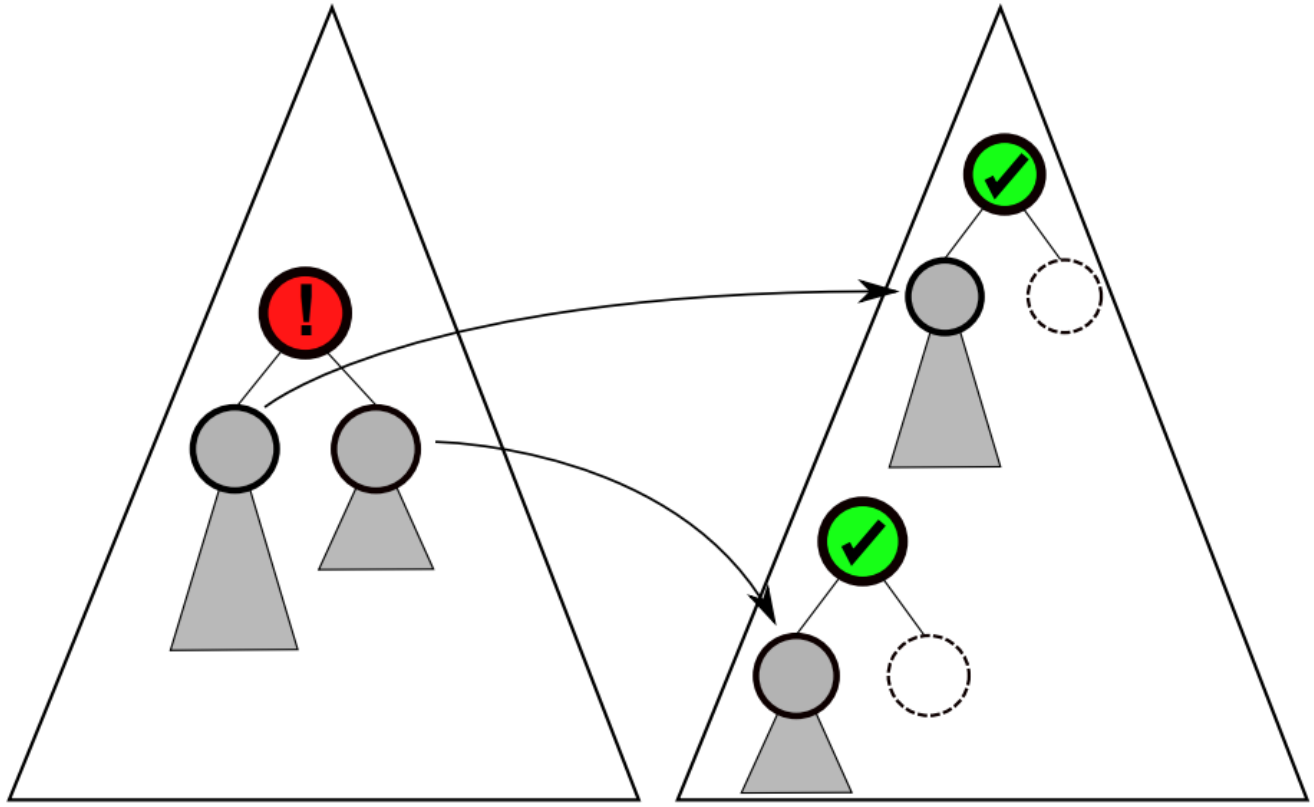
Algoritmus - úvod

- Iterativní zlepšování existujícího BVH do nadstandardní kvality (stavba pomocí SAH)
- Charakterem blízké simulovanému žíhání (Kensler 2008) [2], ovšem globální (oproti lokálním rotacím)
- Nezávislé na zvolené metodě stavby BVH
  - Např. SAH nebo prostorové dělení podle mediánu

Algoritmus

dokud nejsou splněny ukončovací kritéria, dělej:

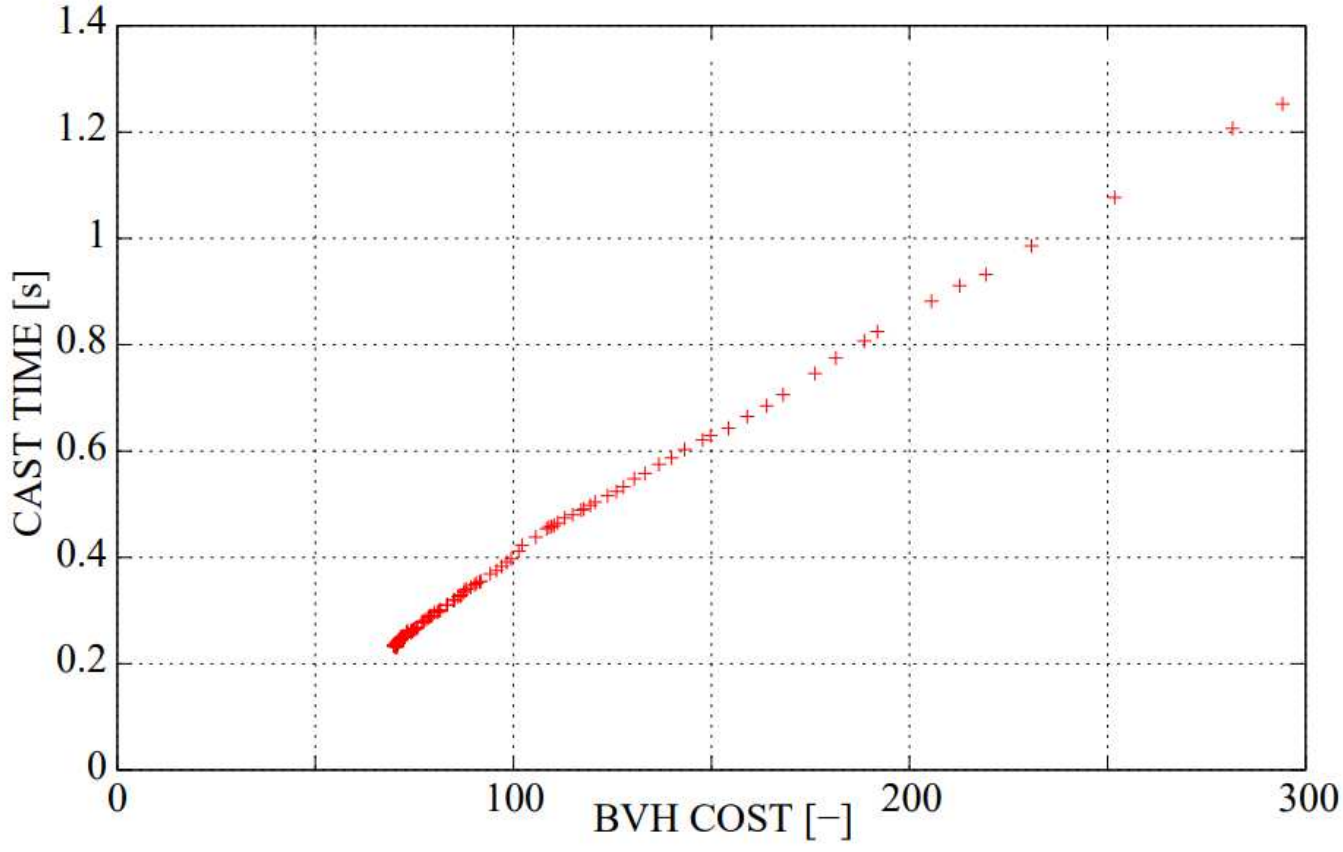
1. zvol vnitřní uzly pro optimalizaci
2. pro každý vnitřní uzel:
   - 2.1 odeber oba jeho potomky ze stromu
   - 2.2 najdi pozici vhodnou pro znovuvložení potomků za použití metody větví a mezí založené na cenách
   - 2.3 vlož každý z potomků na jehou novou pozici a oprav obálky všech afektovaných uzlů



Obrázek 2: Základní myšlenka metody [1].

Cenový model - Surface Area Heuristic (SAH)

- Každý uzel má cenu, která je předpokladem počtu operací pro zpracování daného paprsku
- Předpoklady:
  - Uniformní distribuce paprksů
  - Paprsky nezastíněny - traverzace nekončí po průniku paprsku s libovolným geometrickým primitivem
- Předpoklady v praxi nejsou splněny, ale pro popis ceny je takový model dostačující (podle experimentů)
- $O(N \cdot logN)$ složitost budování, N počet trojúhelníků scény



Obrázek 3: Závislost času vrhání paprsku pro BVH s různou cenou (získaná optimalizací jednoho konkrétního BVH - měřeno na Katedrále sv. Jakuba (Šibenik)) [1].

SAH

Nechť N je zkoumaný uzel. Jeho očekávaná cena je pak dána:

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

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

- Je důležité si všimnout, že pro stromy, kde mohou mít uzly více potomků, se traverzační cena c_T zvyšuje

Cena kořene

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

- Cena kořene reprezentuje očekávaný počet operací pro zpracování paprsku ve scéně
- $SA(T)$ je povrch obálky celé scény
- Všimněte si, že pravá suma je konstantní za předpokladu, že je v listech fixní počet trojúhelníků
  - Tím pádem potřebujeme optimalizovat levou sumu $\rightarrow$ minimalizace součtu povrchů ploch vnitřních uzlů

Aktualizace uzlů

Dělí se na 3 etapy:

1. Odstraňování uzlů
2. Hledání nových pozic pro vyjmuté uzly
3. Zpětné vložení do hierarchie

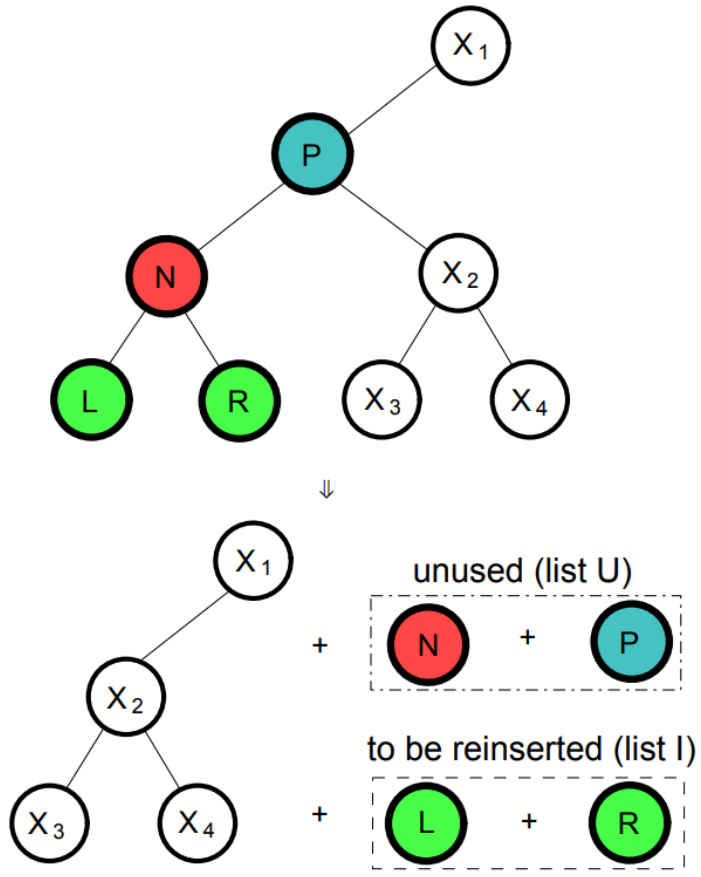
Pro následující slidy:

- Předpokládejme, že jsme zvolili uzel N pro aktualizaci
- Označme levého a pravého potomka N jako L a R
- Označme rodičovský uzel N jako P

Odstranění uzlů

- Při aktualizaci N odstraníme jak N, tak L, R i P a aktualizujeme strom, aby nebyla porušena jeho topologická struktura
- Kromě opravy referencí navíc musíme aktualizovat povrch všech afektovaných uzlů - traverzace až ke kořeni
- Vložíme L a R do uspořádaného seznamu, který obsahuje uzly pro zpětné vložení do hierarchie, 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
- L a R nemusí být listy, ale celé podstromy

Odstranění uzlů



Obrázek 4: Lokální situace před a po odstranění uzlu N [1].

Vyhledávání nových pozic

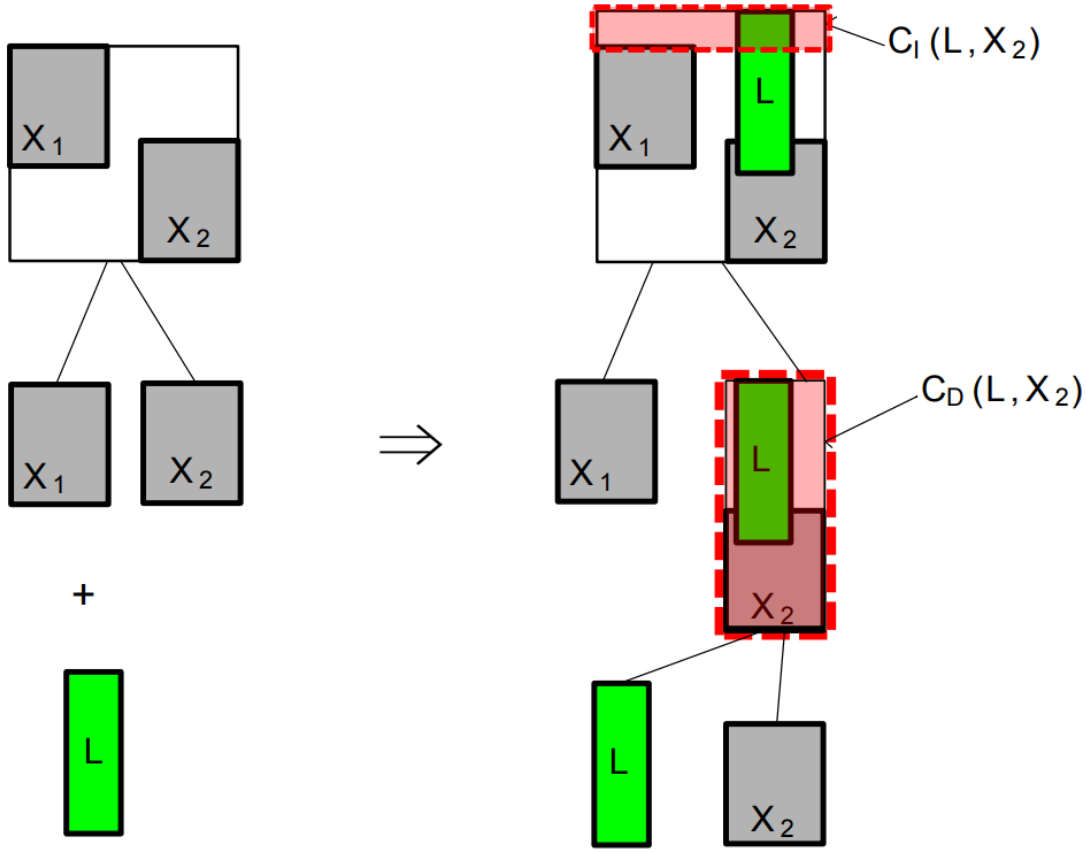
- Hledání nové pozice je tzv. *greedy* (chamtivé) - tzn. že zvolíme pozici, která co nejvíce minimalizuje cenu výsledného BVH
- Začneme v kořeni stromu
- Inkrementálně počítáme zvětšení povrchu ve zkoumaném uzlu (pokud bychom tam vložili uzel ze seznamu uzlů pro znovuvložení)
- Celková hodnota zvětšení povrchu (a tím pádem ceny) je dána dvěmi hodnotami: *přímá* a *indukovaná cena*

Přímá cena: $C_D(L, X) = SA(X \cup L)$

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

Indukovaná cena (induced cost): $C_I(L, X)$

- Akumulované zvýšení povrchu od kořene do rodičovského uzlu právě zkoumaného uzlu X za předpokladu, že by uzel L byl vložen do podstromu, jež má kořen v X
- Rekurzivně: $C_I(L, X) = 0$ pokud je X kořen a $C_I(L, X) = C_I(L, parent(X)) + SA(parent(X) \cup L) - SA(parent(X))$ jinak



Obrázek 5: 2D ukázka přímé ceny C_D a indukované ceny C_I při vkládání uzlu L na pozici uzlu X_2 [1].

- Celkové zvýšení ceny je pak součet těchto dvou hodnot

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

- Hledáme uzel X_{best} takový, který minimalizuje tuto hodnotu pro celý strom

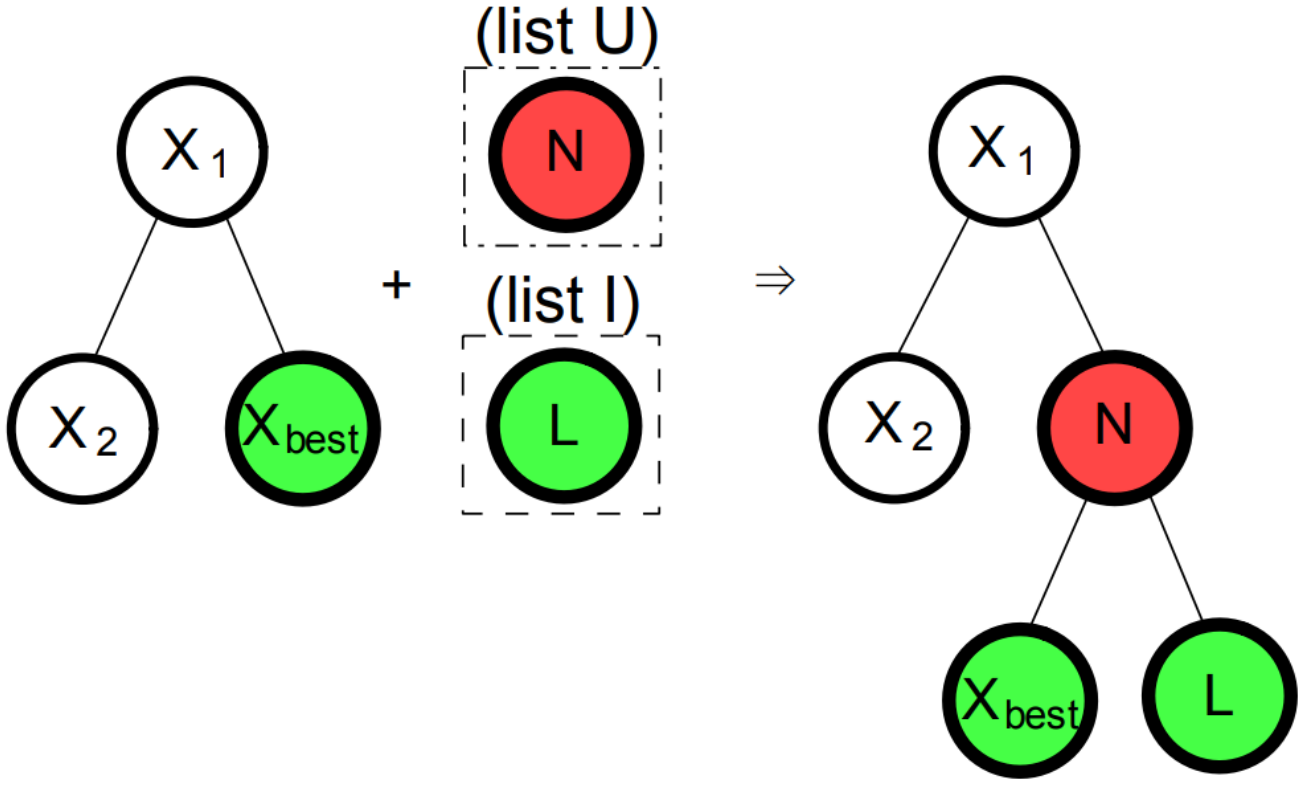
Metoda větví a mezí (Branch and Bound Method)

- Založena na prioritní frontě, kde priorita je inverzně proporcionální indukované ceně
- Můžeme vyhledávání ořezávat/ukončovat na základě současně nejmenší hodnoty C_{best} korespondující s X_{best}
- Pro ořezání musíme zjistit dolní mez (lower bound) v podstromě X abychom rozhodli, zda tam má cenu pokračovat
  - Ta je dána indukovanou cenou nad X a povrchem L
  - Ind. cena nad X reprezentuje nutné zvětšení uzlů nad X
  - Povrch L je minimální velikost uzlu vloženého do stromu
- Celý algoritmus může být ukončen v případě, kdy je dolní mez prvního uzlu v prioritní frontě větší jak 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;
```

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 rodiče
- Po každém vložení musíme aktualizovat povrchy všech obálek od společného rodiče ke kořeni



Obrázek 6: Ilustrace situace, kdy vkládáme zpět uzel L na pozici X_{best} do stromu hierarchie. Zde za použití N jako společného rodiče [1].

Volba uzlů k aktualizaci

Volba uzlů - úvod

- Náhodný výběr (random sampling)
  - Redukce ceny BVH dokud nedokonvergujeme
- Pro urychlení, je nutné zvolit chytřejší způsob
- Zavedeme tzv. hodnotu neefektivnosti uzlu (node inefficiency measure)
- Potřebujeme, aby korelovala s opravdovým snížením celkové ceny stromu pro daný uzel

Možné strategie volby uzlů

M_{SUM}

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

- $|children\ of\ N|$ je počet potomků uzlu N
- Estimuje relativní zvýšení povrchu uzlu N vůči potomkům
- Pokud má N velkou obálku zatímco jeho potomci je mají malé a tím pádem je v N mnoho prázdného místa, pak hodnota M_{SUM} bude veliká

M_{MIN}

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

- $min_{X \in children\ of\ N} SA(X)$ je minimum z povrchů potomků N
- Vhodné na situace, kdy je vysoký rozdíl mezi velikostmi potomků
- Např. jeden potomek je terén scény, zatímco ten druhý je automobil
- M_{SUM} není schopno tuto situaci řešit/rozpoznat, neboť součet a tím pádem i průměr povrchů potomků zůstane dostatečně velký

M_{AREA}

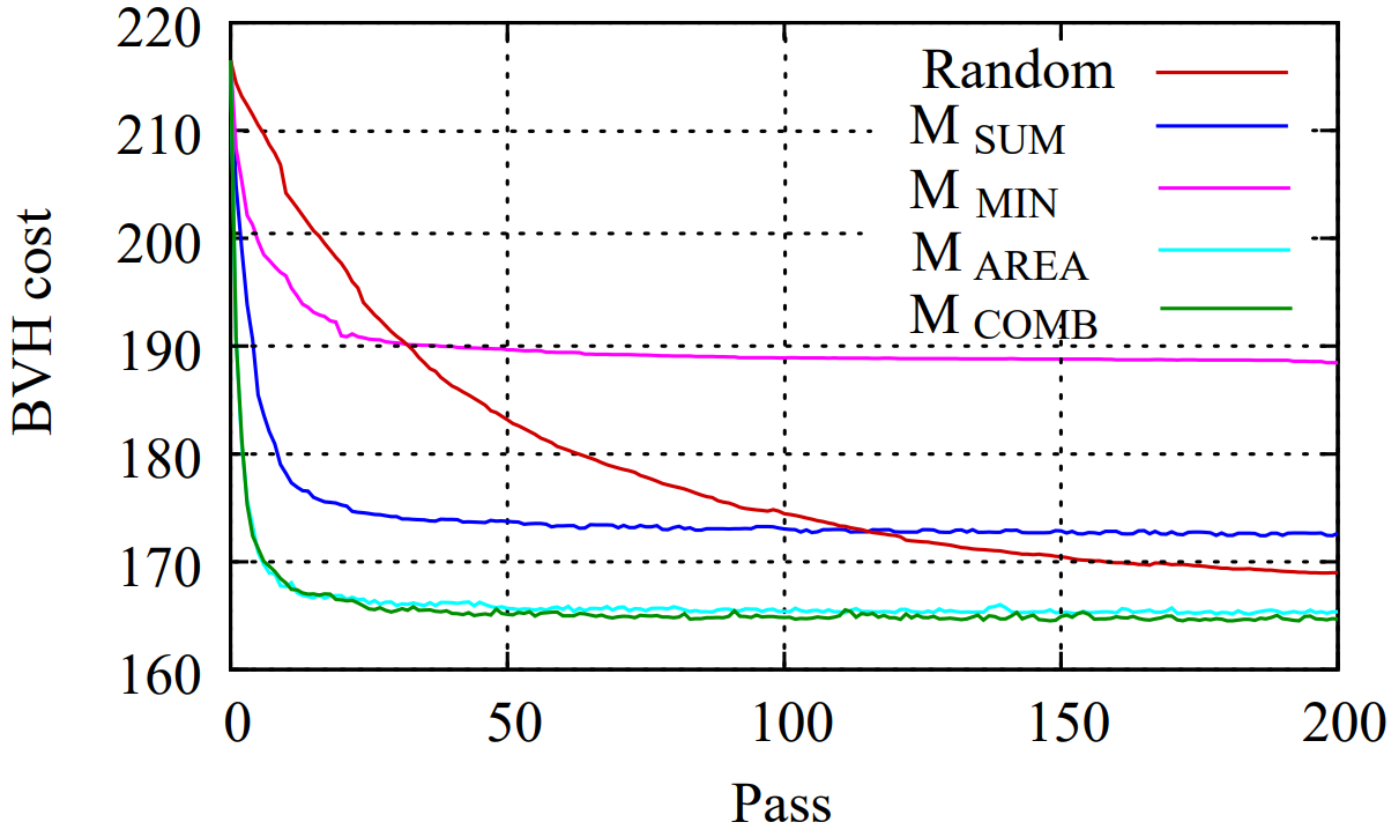
$$M_{AREA}(N) = SA(N)$$

- Prioritizuje vylepšování uzlů s velikým povrchem plochy

Kombinace všech strategií

- M_{SUM}, M_{MIN} a M_{AREA} jsou schopny detekovat mnoho situací, ve kterých může dojít po volbě a vylepšení uzlu ke značnému zlepšení globální ceny hierarchie
- Podle experimentálních výsledků uvedených v článku dochází k nejrychlejšímu snížení celkové ceny za použití součinu těchto hodnot

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



Obrázek 7: Jednotlivé strategie a zlepšení BVH v počtu průchodů [1].

Kdy a kolik toho aktualizovat?

- Metoda popsaná v článku pracuje na základě iterativního vylepšování k uzlů (typicky $k = 1\%$ všech uzlů)
- Nejdříve je vypočítána neefektivnost všech vnitřních uzlů
- Poté zvolíme k těch nejméně efektivních
- Ty pak sekvenčně zpracujeme od nejméně efektivního

Možnost aktualizace jen toho nejméně efektivního uzlu

- Je možné iterativně aktualizovat vždy ten nejméně efektivní uzel
- ale:
  - Zpracování skupiny uzlů (batch processing) je rychlejší, neboť spočítáme hodnotu neefektivity pro každých $k\%$ uzlů namísto jednoho v jedné iteraci
  - Navíc je zpracování skupiny uzlů více robustní v rámci zaseknutí se v lokálním minimu (opakované aktualizaci problematického uzlu)

Redukce celkové ceny

- Během počátečních iterací dojde k redukci celkové ceny ve většině případech
- Může ale dojít k situaci, kdy po vložení obou vyjmutých potomků dojde ke zvýšení ceny
  - Tento jev je především značný v situaci, kdy optimalizace osciluje v oblasti minima a cena BVH již nelze jednoduše redukovat
- Modifikací algoritmu, kde je vyjmut pouze jeden potomek uzlu, lze zaručit, že bude celková cena vždy snížena nebo zůstane stejná
  - Zbytečné, neboť redukce je rychlejší pro metodu, kde jsou vždy odebráni oba potomci

Ukončení algoritmu

- Více možností jak rozhodnout, zda má alg. skončit
  - Např. podle uplynulého času nebo počtu průchodů
- Možnost evaluace konvergence ceny
- Podle autorů článku: ukončit algoritmus, pokud se cena nezlepší během daného počtu průchodů p_T
- Za předpokladu, že je používán výběr uzlů na základě jejich neefektivity, celková cena může dosáhnout o trochu vyšších hodnot jak náhodný výběr, neboť jsou uzly, které nemusí být nikdy vybrány pro aktualizaci

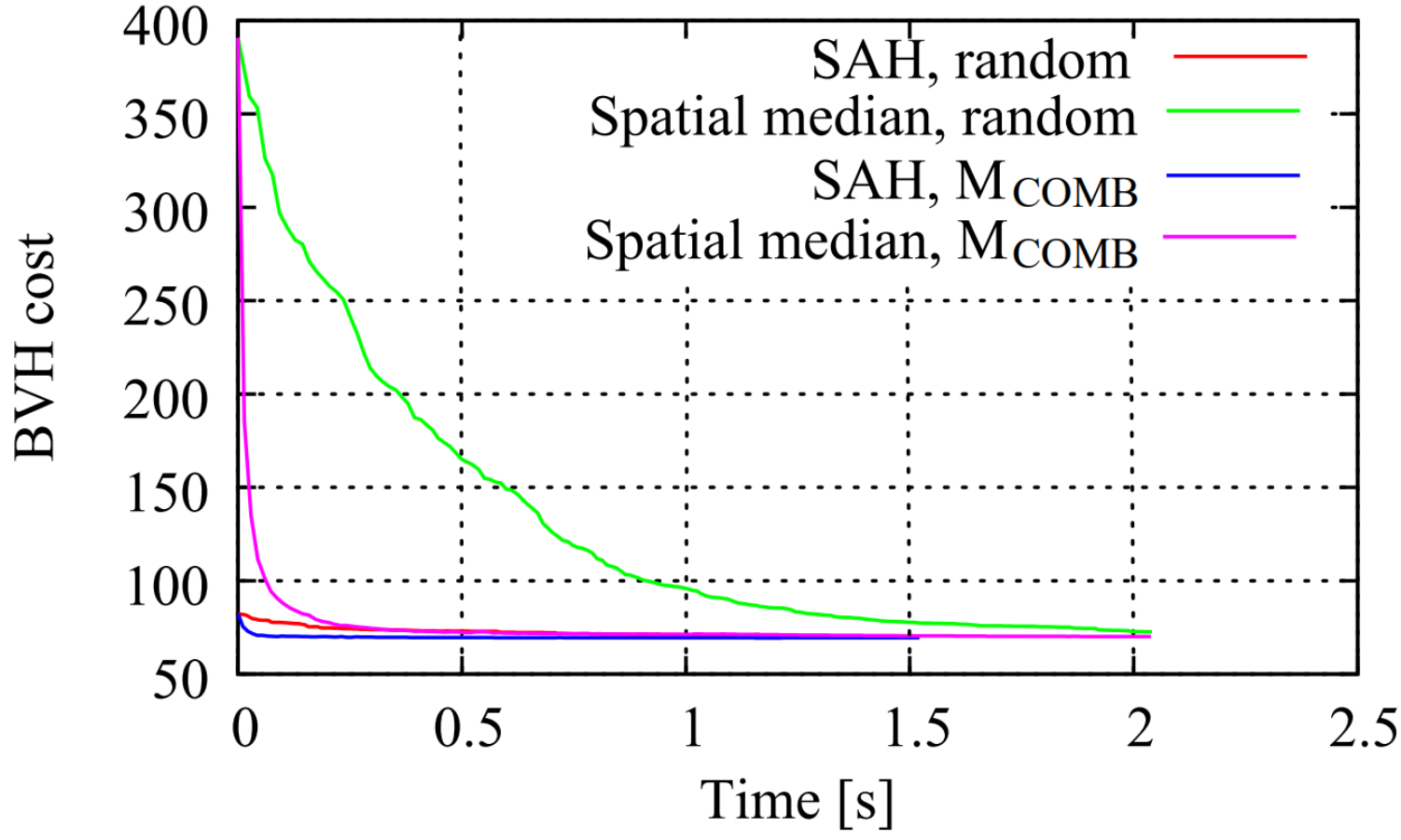
- Abychom zamezili tomuto chování, v situaci, kdy snížení ceny je nízké nebo nulové, přepneme na náhodný výběr
- Podobně jako pro celkový algoritmus, toto rozhodnutí platí pro skupiny uzlů (vždy vylepšujeme k uzlů)
- Pro přepnutí z kombinované neefektivity na náhodný výběr zavedeme proměnnou p_R ($p_R \leq p_T$)

Počet trojúhelníků v listech

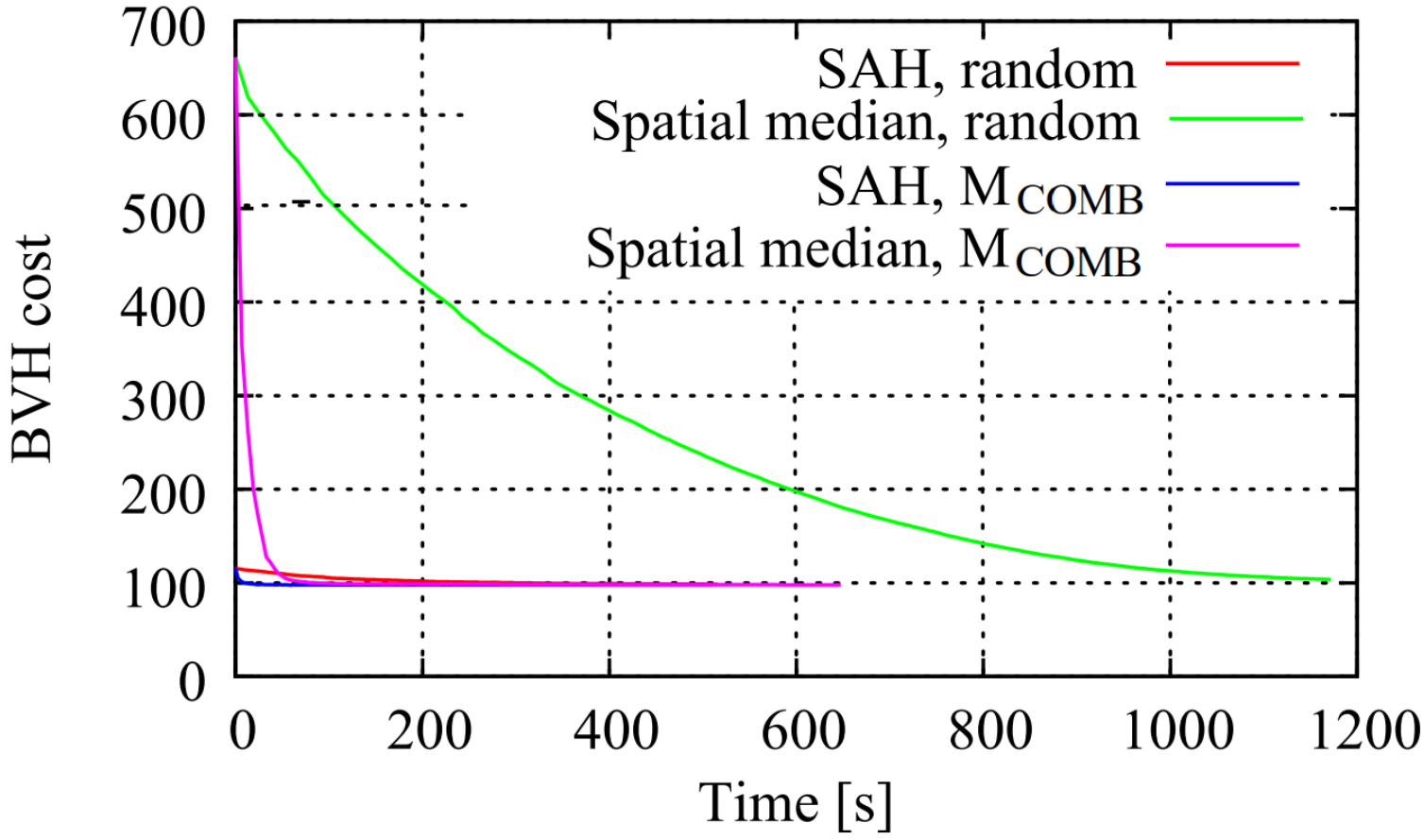
- Předpoklad: počet trojúhelníků v každém listu je roven jedné (nebo obecně geom. primitivum)
- V mnoha případech je výhodnější mít více trojúhelníků v listech (např. 8 až 10)
- Možné rozdělení na dvě etapy:
  1. BVH s jedním trojúhelníkem v listech + jeho optimalizace
  2. Postprocessing $\rightarrow$ post-order průchod stromem, evaluace $C(N)$ $\forall N$, musíme též spočítat počet trojúhelníků asociovaných s uzlem N
     - Pokud je cena vnitřního uzlu N vyšší jak cena uzlu vytvořeného ze sloučení trojúhelníků z listů v jeho podstromech v jeden list, pak sloučíme podstromy N v jeden list
     - Časová náročnost postprocess kroku je lineární vzhledem k počtu uzlů hierarchie

Výsledky

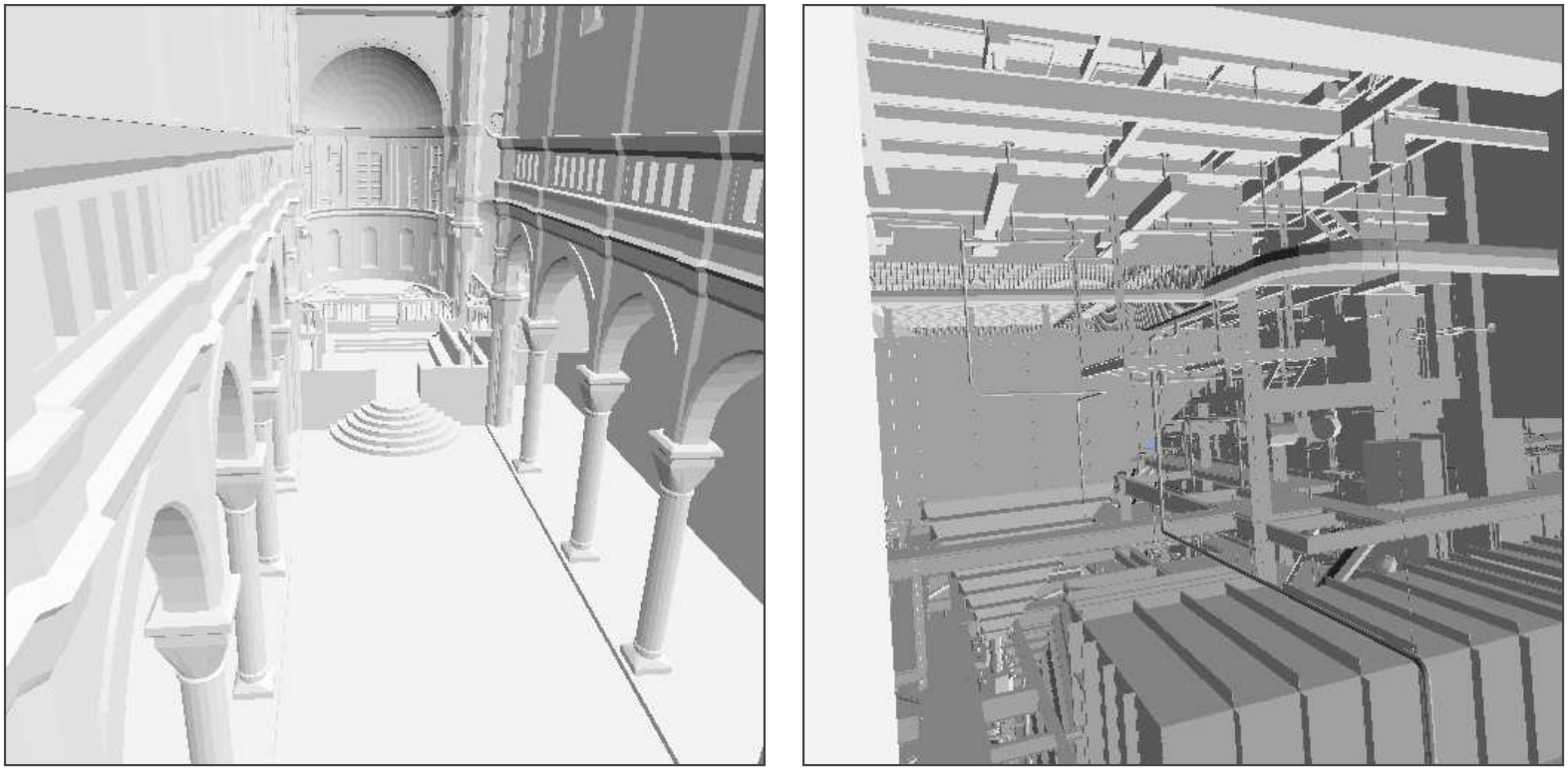
- Testováno na čtrnácti scénách různé komplexity
  - S jedním složitým objektem
  - Architektonické scény
- Až 27% zlepšení pro BVH původně vytvořené za pomocí SAH
- Až 88% zlepšení pro BVH původně vytvořené za použití prostorového dělení v mediánu



Obrázek 8: Porovnání metody použité na BVH vybudovaných za pomocí SAH a prostorového mediánu - náhodná a M_{COMB} volba uzlů k aktualizaci (na Katedrále sv. Jakuba (Šibenik) - 80 tis. troj.) [1].



Obrázek 9: Porovnání metody použité na BVH vybudovaných za pomocí SAH a prostorového mediánu - náhodná a M_{COMB} volba uzlů k aktualizaci (na scéně elektrárny - 12 748 tis. troj.) [1].



(a) (b)

Obrázek 10: Vytvořené snímky.
(a) Katedrála sv. Jakuba (Šibenik)
(b) Elektrárna

Porovnání se simulovaným žíháním

- Volba uzlů podle priorit (v prioritní frontě) vs. prohledávání všech uzlů do hloubky
- Globální změny (aktualizace jednoho uzlu může vést k velké redukci ceny) vs. lokální rotace, které postupně ale jistě snižují celkovou cenu
- Optimalizace BVH 10-25x rychlejší jak simulované žíhání pro individuální objekty
- Až o 10% nižší cena s 28-80x rychlejší stavbou jak simulované žíhání pro architektonické scény

Scene	cost[-] SAH build	cost, our optimized SAH build	cost, spatial median	cost, our optimized sp. median	time[s], SAH build	time, our optimized SAH build	time, spatial median	time, our optimized sp. median
Conference	130.30 (100%)	78.66%	646.38%	78.83%	3.45 (100%)	131.30%	12.75%	226.38%
Fairy Forest	95.10 (100%)	96.21%	194.52%	97.70%	1.96 (100%)	120.92%	12.24%	63.27%
Sibenik Cathedral	82.30 (100%)	84.45%	474.74%	86.60%	0.66 (100%)	209.09%	13.64%	175.76%
Sponza	220.20 (100%)	82.74%	571.25%	83.30%	0.56 (100%)	180.36%	12.50%	376.79%
Soda Hall	216.50 (100%)	76.40%	644.96%	76.61%	40.96 (100%)	137.60%	10.47%	187.26%
Power Plant, sec. 9	57.90 (100%)	89.46%	325.87%	90.33%	1.36 (100%)	119.12%	11.76%	205.15%
Power Plant, sec. 16	93.50 (100%)	85.35%	540.16%	85.24%	4.70 (100%)	159.57%	12.77%	378.51%
Power Plant	115.80 (100%)	84.46%	571.10%	85.16%	396.00 (100%)	121.46%	9.08%	71.90%
Pompeii Ten	252.90 (100%)	86.20%	303.24%	86.98%	102.00 (100%)	175.49%	10.76%	114.02%

Obrázek 11: Souhrn výsledků při použití metody na architektonické scény, kde je referenční hodnotou BVH vybudované za pomoci SAH bez jakékoliv další optimalizace (100%) [1].

Literatura

[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

Děkuji za
pozornost