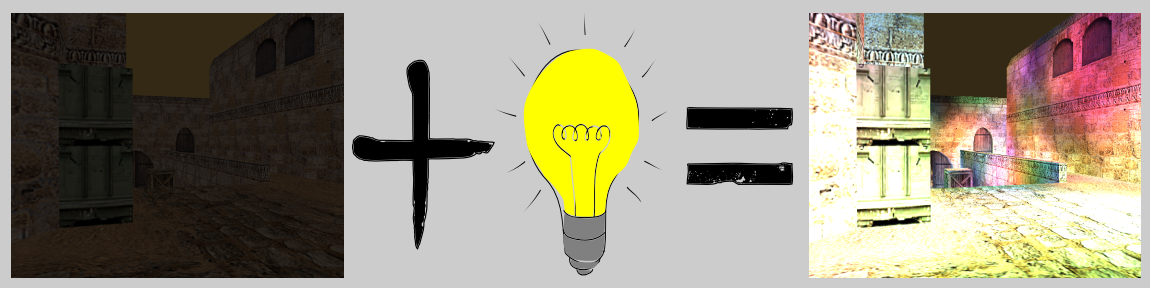
BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2019
http://excel.fit.vutbr.cz

Clustered deferred shading vo Vulkan API

Matej Karas*



Abstrakt

Táto práca sa zaoberá metódou vykresľovania viacerých svetiel v reálnom čase. Výpočet osvetlenia je drahá operácia a pri naivných technikách (ktoré síce postačujú pre aplikácie s nízkym počtom svetiel), často dochádza k redundantným výpočtom. Cieľom práce je implementovať metódu clustered deferred shading v aplikačnom rozhraní Vulkan a efektívne vykresľovať viacero svetiel v reálnom čase.

V terajšej implementácii sa podarilo efektívne zredukovať počet svetelných kalkulácií a tým urýchliť vykresľovanie resp. zvýšiť FPS a to až 100-násobne pri 10k a viac svetlách v scéne voči klasickému deferred shadingu.

Kľúčové slová: Vulkan, Clustered shading, Deferred rendering, Graphics, Many lights, Real–time rendering

Priložené materiály: Demonštračné video

*xkaras34@fit.vutbr.cz, *Fakulta Informačních Technologií, Vysoké Učení Technické v Brně*

1. Introduction

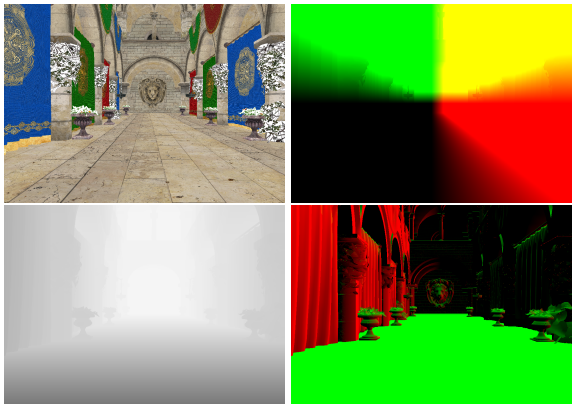
Výpočet osvetlenia je náročná operácia a spotrebuje veľa výpočetných zdrojov. Pri naivných metódach vykresľovania (brute-force výpočet všetkých svetiel, ktoré sú v scéne), dochádza k mnoho redundantným výpočtom. Aby sa zredukovali tieto redundantné výpočty, je potrebné odlíšiť svetlá, ktoré nebudú pri vykresľovaní daného fragmentu potrebné. Na dosiahnutie je nutné vytvoriť akceleračnú štruktúru, ktorá umožní jednoduchý priechod svetlami.

2. Predchádzajúca práca

Clustered shading [1] vychádza z predchádzajúcej techniky, *tiled shading* [2]. *Tiled shading* pracuje spôsobom ukladania svetieľ v screen–space tiles (dlaždíc). Každá tile vytvára sub-frustum na základe min–max hĺbky a obsahuje list svetiel, ktoré ju potenciálne zasahujú. Pokiaľ sú v tile objekty s rôznou hĺbkou, tak v najhoršom prípade test subfrusta zdegeneruje na obyčajný 2D test, a redukcia redundantných svetiel je minimálna.

Tiled shading má podobnosti so starou technikou *Tiled Rendering* [3], v ktorej je delenie do tiles aplikované na objekty, a nie svetlá.

Clustered shading rieši problémy predchádzajäcej metódy tým, že nepoužíva 2D tiles, ale 3D clustre a tým odstraňuje hĺbkové nesúvislosti a zaručuje dobrú redukciu svetiel v reálnom čase. Taktiež list svetiel, ktoré potenciálne zasahujú cluster, je menší než pri tiled shading. Pre ukladanie clustrov sú použité virtuálne tabuľky stránok, pretože počet všetkých možných clustrov by sa nezmestil do pamäte, resp. pri dnešných grafických kartách by zaberal príliš veľa pamäte.



Obrázok 1. Vizualizácia uloženia jednotlivých komponentov v G-bufferoch. Aby sa šetrila pamäť, sú niektoré komponenty, napr. spekulárna zložka svetla, pribalené do buffera k iným komponentom.

3. Deferred rendering

Deferred rendering je technika, ako už názov napovedá, pri ktorej sa geometria nevykresľuje priamo, ale odkladá sa do tzv. geometry bufferov (G-bufferov) a je použitá neskôr pri výpočte osvetlenia. Spôsob uloženia jednotlivých zložiek tak, aby sa šetrila pamäť, je zobrazený na obrázku 1.

Výhodou tejto techniky je, že potrebujeme iba jeden priechod geometriou. Osvetlenie sa počíta neskôr a pre jeden fragment práve jedenkrát. Nevýhodou je, že nedokážeme vykresľovať transparentné objekty a používanie G-bufferov vedie na značne väčšie nároky na priepustnosť pamäte.

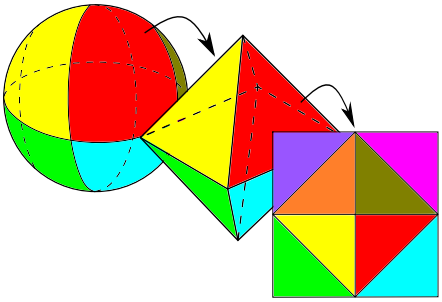
Pri vytváraní G-bufferov sú normály a pozície transformované do view-space, čo umožňuje 2 optimalizácie a to:

- Pri výpočte osvetlenia sa nemusí počítať s kamerou (pretože tá sa nachádza v počiatku)
- Kompresia normál (3.1)

3.1 Kompresia normál

Na kompresiu normál je použitá technika mapovania gule na octahedron a následne premietnutie na z-rovinu a reflektovanie –z-pologule cez príslušnú diagonálu. Táto metóda bola jedna z najefektívnejších a s najmenšou chybou podľa [4]. Dôvod, prečo je táto metóda výpočtovo jednoduchá, je, že mapovanie gule na octahedron je obyčajná zmena definície vzdialenosti z Euklidovskej do Manhattenskej podľa rovnice 1. Celý proces mapovania je naznačený na obrázku 2.

$$m(x,y) = \frac{e.xy}{|e.x| + |e.y| + |e.z|} \tag{1}$$



Obrázok 2. Mapovanie gule na octahedron a následne jej premietnutie na rovinu prebieha tak, že najskôr sú premietnuté osminy z gule na strany octahedronu, z ktorého sú potom priemetnuté na rovinu.

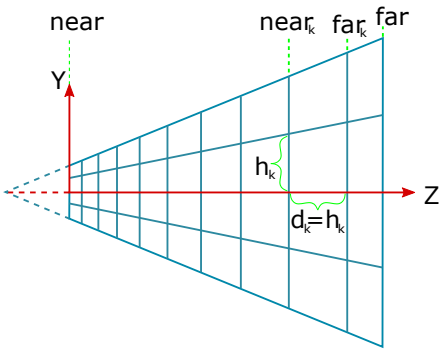
4. Clustered shading

Clustered shading je technika, v ktorej je view-frustum rozdelené na menšie sub-frustá na základe pozície fragmentu v screen-space a hĺbky daného fragmentu, ktoré sa získajú z G-bufferov. Delenie pozdĺž z-osi je exponenciálne, pretože pri lineárnom delení by clustre (zhluky) blízko near plane zostali tenké a naopak, pri far plane dlhé. Exponenciálne delenie zaručí, že clustre budú mať približne rovnakú veľkosť (obr. 3). [1]

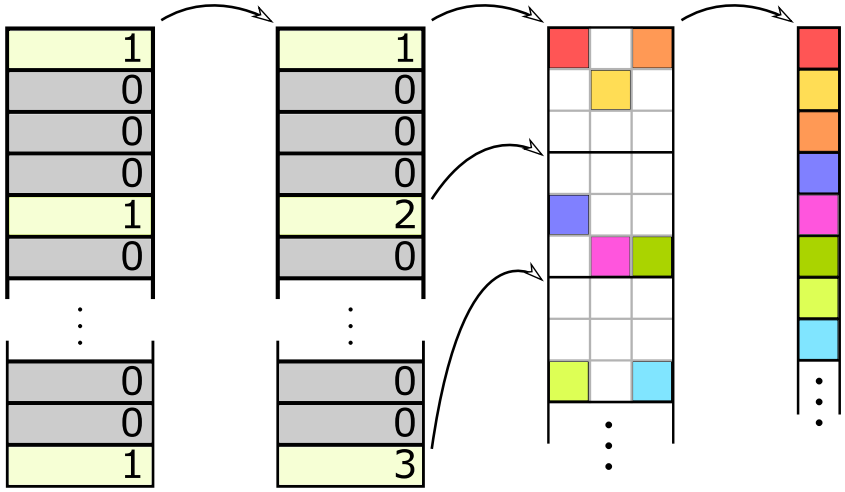
Cluster, resp. subfrustum sa identifikuje podľa trojice $c = (i, j, k)$, kde koordináty (i, j) sú indexy screen-space tiles, ktoré sú vypočítané ako $(i, j) = (\lfloor x_{ss}/t_x \rfloor, \lfloor y_{ss}/t_y \rfloor)$, pre veľkosť tile (t_x, t_y). Index k sa vypočíta podľa rovnice 2, ktorá vykonáva exponenciálne delenie pozdĺž z-osi vo view-space. [1]

$$k = \left\lfloor \frac{\log\left(-z_{vs}/near\right)}{\log\left(1 + \frac{2\tan\theta}{S_y}\right)} \right\rfloor \tag{2}$$

Trojica (i, j, k) sa zkompaktuje do 32-bitov tak, že pre indexy i a j postačuje 7bitov (pri veľkosti $t = 32$ pixelov) na 4K rozlíšenie a zvyšné bity pre uchovanie indexu k.



Obrázok 3. Exponenciálne delenie view-frusta na subfrustá. Z obrázku je vidieť, že exponenciálne delenie zaručí približne rovnaké rozmery clustrov pri danej k-partícii.



Obrázok 4. Jednotlivé kroky vykonávane s tabuľkou stránok, ktoré vedú na vytvorenie unikátnich clustrov. **Vľavo** – Označenie potrebných stránok v tabuľke. Na miesto alokácie sa zapíše značka. Tabuľka je v počiatočnom stave vynulovaná. **Stredný zľava** – Alokácia potrebných stránok. Značky indikujúce potrebnú alokáciu sa prepíšu unikátnym indexom, začínajúcim od 1. **Stredný zprava** – Uloženie clustrov do alokovaných stránok. Virtuálna adresa clustru sa preloží na adresu globálneho buffera stránok a na túto adresu sa uloží daný cluster. **Vpravo** – Kompakcia clustrov a vytvorenie zoznamu unikátnych clustrov. Clustre v globálnom bufferi stránok sú už čiastočne zkompaktované. Po ich zkompaktovaní vznikne list unikátnych clustrov, ktoré sa zapíšu do globálnej pamäte.

5. Unikátne clustre

Po priradení fragmentov do clustrov sa musia určiť unikátne clustre, aby sa zbitočne nehľadal list zasahujúcich svetiel pre rovnaké clustre. Na to, aby sa dokázali určiť unikátne clustre, treba list clustrov skompaktovať.

Na kompakciu som použil virtuálne tabuľky stránok, pretože sa ukázali efektívnejšie ako triedenie [1], ktoré je aj v dnešnej dobe stále drahá operácia (obr. 8). Algoritmus tabuliek je následovný:

1. Označenie potrebných stránok v tabuľke
2. Alokácia stránok
3. Uloženie clustrov do stránok
4. Kompakcia stránok

Označenie stránok je obyčajné zapísanie značky do tabuľky, ktorá bude indikovať nutnosť alokácie danej stránky (obr. 4). Miesto zapísania značky pri veľkosti jednej stránky $s = 2^9$, sa určí ako cluster c/s. Z toho vyplýva, že pri veľkosti 32 bitov na jeden záznam v tabuľke, stránku určuje vrchných 23 bitov skompaktovaného kľúča. A teda veľkosť tabuľky musí byť 2^{23}, resp. 32MB. Avšak pri mojich parametroch, index k vyžaduje len 7bitov, a teda veľkosť tabuľky musí byť 2^{12} a celá tabuľka bude v pamäti zaberať 16kB.

Alokácia stránok predstavuje zapísanie unikátneho indexu na miesto značky, ktorý bude odkazovať do globálneho bufferu stránok (obr. 4), ktorý slúži ako ukazateľ stránky. Ukazateľ na začiatok stránky S sa dostane ako $S = index \cdot 2^9$.

Adresa uloženia daného clustera sa dostane podľa pseudo kódu 1 [5] a skompaktovaný kľúč clustru sa uloží v globálnom buffery stránok na pozíciu adresy získanú z prekladu (obr. 4). Tak ako pri kroku 5 sa opäť prejdú všetky fragmenty a popísaným spôsobom sa uložia všetky zasiahnuté clustre. Kroky 1 až 3 by sa mohli zlúčiť do jedného priechodu, avšak podľa [1] to vedie na 2-násobné spomalnie, ale stále predstavuje možné riešenie na hardware s rýchlejšími atomickými operáciami.

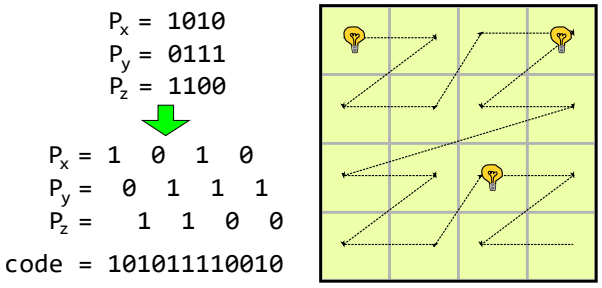
```
pageNum = virtualAddress >> 9
pageAddr = (nodes[pageNum] - 1) * 512
offset = virtualAddress % 512
return pageAddr + offset
```

Listing 1. Pseudo kód prekladu virtuálnej adresy na fyzickú pomocou jedno-úrovňovej tabuľky stránok. Funkcia vráti adresu v globálnom bufferi stránok, do ktorej sa uloží daný cluster.

Pri kompakcii je využitý fakt, že stránky sú už skompaktované v globálnom bufferi stránok, avšak nie všetky prvky sú v jednotlivých stránkach využité (obr. 4). Po skompaktovaní stránok vznikne list unikátnych clustrov, ktorý sa zapíše do globálnej pamäte.

5.1 Priradenie svetiel

Naivný postup priradenia svetiel by predstavoval pre každý cluster prejsť všetky svetlá a zistiť, ktoré ho ovplyvňujú. Tento spôsob je postačujúci pre menší



Obrázok 5. Znázornenie Mortonovej krivky pre 2D body. Mortonova krivka mapuje súradnice bodu do jednej dimenzie postupným prekladaním jednotlivých bitov súradníc daného bodu.

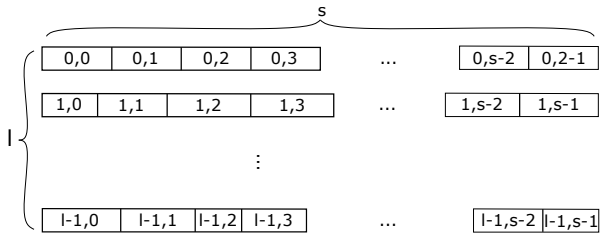
počet, avšak pri väčšom počte svetiel je pokles výkonu znateľný. Z tohoto dôvodu treba list svetiel usporiadať tak, aby bolo možné jednoducho zistiť, ktoré svetlá ovplyvňujú jednotlivé clustre. Keďže by nebolo jednoduché zoradiť 3D pozície svetiel, je nutné ich namapovať na 1D, kde bude triedenie jednoduché. Na mapovanie je použitá mortonova krivka 5.2, ktorej výpočet je jednoduchší oproti iným mapovacím algoritmom. Po zotriedení svetiel sa uložia do Bounding Volume Hierarchy (BVH) stromu, ktorý umožní jednoduchý priechod listom svetiel 6.2.

5.2 Mortonova krivka

Mortonova krivka je spôsob, ktorým je možné mapovať N-dimenzionálny priestor do jednej dimenzie. Mortonov kód svetla je získaný tak, že jednotlivé bity súradníc jeho pozície sa preložia tak, ako je znázornené na obrázku 5.

5.3 Vykresľovanie

Vykresľovanie sa líši voči klasickému deferred shadingu len v tom, ako sa získa list svetiel. Pri klasickom resp. naivnom vykresľovaní, sa zoberie list všetkých svetiel a tie sa testujú voči danému fragmentu. V mojej metóde je nutné najprv získať kľúč clustru, v ktorom sa fragment nachádza a následne list svetiel, ktoré zasahujú daný cluster. Kľúč clustra sa získa tým istým spôsobom ako v 4. S kľúčom je možné vyhľadať list svetiel, ktorý daný cluster zasahuje tak, ako bolo popísané v 6.2. Po získaní listu svetiel je výpočet osvetlenia rovnaký ako v klasickom deferred shadingu. Doba jednotlivých krokov je znázornená na obrázku 7 a je z neho vidieť, že metóda je spomalovaná nekvalitným radiacim algoritmom. Vzorky boli získané zo scény, v ktorej bola vysoká hustota svetiel na jeden cluster (až 900 pri 150k svetiel) a z toho dôvodu trvá vykreslovanie tak dlho.



Obrázok 6. Rozdelenie veľkých listov l na menšie sublisty s podľa splitterov. Splittre sa získajú náhodným vzorkovaním nezotriedenej sekvencie a následným vybratím s vzorkov.

6. BVH strom

BVH strom umožní ľahký a rýchli priechod svetlami a zrýchli algoritmus priraďovania svetiel. Na vytvorenie stromu je nutné najskôr list svetiel zoradiť. Vytvorenie celého stromu je v asynchrónnej fronte, pokiaľ ju daný HW podporuje.

6.1 Bitonic-warp-merge sort

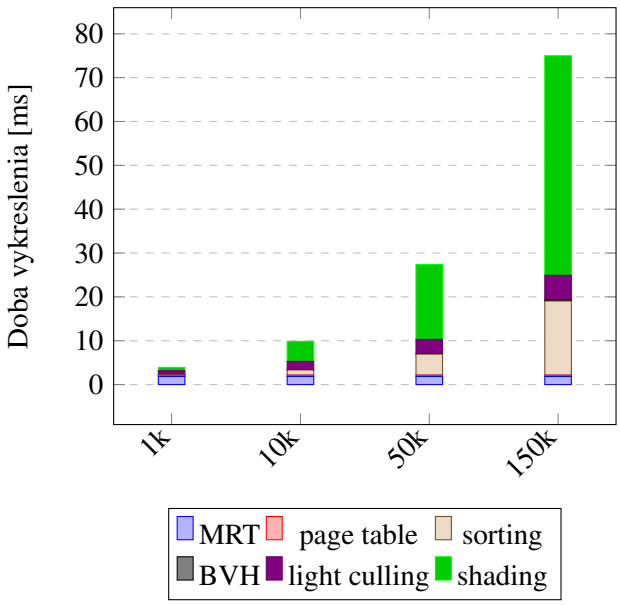
Na zoradenie svetiel je využitý Bitonic-warp merge sort, ktorý sa ukázal byť ako jeden z najrýchlejších porovnávacích sortov. [6] Bitonic-warp sort je bezbarierový sort, a teda každý warp[1] triedi priradené prvky nezávisle od ostatných, v čom spočíva jeho efektivita oproti ostatným porovnávacím radiacim algoritmom. Aby sa zakryla pamäťová latencia a zvýšila efektivita, sa každému vláknu priradia 4 prvky a teda jeden warp zoradí 128 elementov.

Po prvotnom zoradení ostanú 128-prvkové listy, ktoré sa budú postupne spájať do väčších listov. Každý warp obdrží 2 listy A *a* B, z ktorých zoberie 64 najmenších elementov a uloží maximálny element a_{max} a b_{max} z oboch listov. Vhodným usporiadaním sublistov A *a* B bude na ich zoradenie postačovať posledný priechod bitonickej siete. Po zoradení sa najmenších zoradených 64 prvkov uloží do globálnej pamäte. Potom sa zoberie ďalších 64 elementov z listu, ktorého maximálny element bol väčší a uloží sa jeho nový maximálny element, až pokiaľ nie sú zlúčené oba listy.

Vzniknuté listy sa budú zlučovať tým istým spôsobom, pokiaľ postačuje paralelizmus, pretože každým zlučovaním sa geometricky znižuje počet listov na zlučovanie. Keď už paralelizmus nestačí, každý zvyšný list l sa rozdelí podľa splitterov na s sub-listov tak, ako je naznačené na obrázku 6. Splittre sa získali tak, že počiatočná nezotriedená sekvencia Mortonových kódov sa náhodne navzorkuje tak, aby vzniklo $s \cdot k$ vzorkov, ktoré sa zotriedia a následne sa z nich vyberie každý k-ty prvok.

Takto vzniknuté sublisty je opäť možné triediť paralelne. Aby vznikol konečný zotriedený list, je nutné

[1] Basic Concepts in GPU Computing



Obrázok 7. Doba jednotlivých krokov. Z hodnôt vyplíva, že vykreslovanie je spomalované nekvalitným radiacim algoritmom. Je nutne podotknúť, že v scéne bola vysoká hustota svetiel na jeden cluster, a z toho dôvodu trvá vykreslovanie tak dlho.

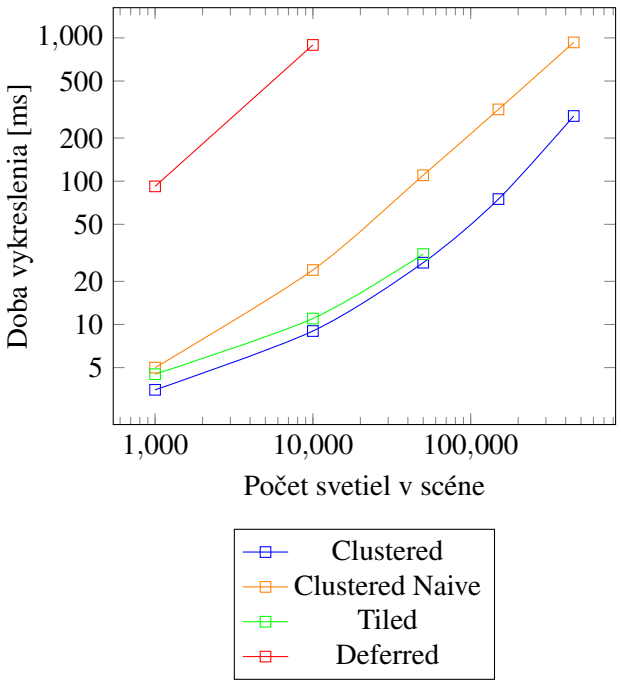
zlúčiť sublisty $(0,i),(1,i),\ldots,(l-1,i)$ do listov S_i. Pretože pre vzniknuté listy S_i platí vzťah 3, je možné ich jednoducho usporiadať do finálne zoradeného listu.

$$\forall a \in S_i, \forall b \in S_j, i < j : a < b \tag{3}$$

6.2 BVH

Zoradené svetlá sa uložia do BVH stromu, ktorý bude umožňovať rýchle priradenie svetiel ku clustrom. Listy stromu sa získajú priamo zo zoradených svetiel. Na získanie ďalšej úrovne je zlúčených 32 alebo 64, bezprostredných listov do obaľovej plochy (AABB). Takto sa postupuje, pokiaľ nezostane posledná úroveň obsahujúca maximálne veľkosť vetviaceho faktoru prvkov. Vetviaci faktor 32 alebo 64 poskytuje efektívny priechod stromom pri Nvidia resp. AMD kartách, pretože o každý priechod sa postará jeden warp a zároveň ostane strom plytký.

Po vytvorení stromu sa priechodom do hĺbky vytvorí pre každý unikátny cluster list svetiel, ktoré ho zasahujú. Na každej úrovni sa testuje AABB voči subfrustu daného clustra. Pre listy sa použije obalový tvar daného svetla. Do globálnej pamäte sa zapíše počet svetiel, ktorý daný list obsahuje a následne indexy svetiel, ktoré ho zasahujú. Do tabuľky stránok sa uloží na miesto kľúča clustru adresa, kde je uložený daný list. Potom virtuálne vyhľadanie kľúča v tabuľke, vráti list svetiel zasahujúcich daný cluster.



Obrázok 8. Rýchlosť vykresľovania pri jednotlivých technikách a počtoch svetiel na karte Nvidia GTX 960m pri rozlíšení 1024×726. Moja implementácia tiled shadingu dovoľuje 1024 svetiel pre jednu tile, a preto pri 50k a viac svetiel začali byť tile saturované a výsledky boli skreslené. Pri klasickom deferred vykresľovaní bolo možné vykresliť maximálne 10k svetiel.

7. Porovnanie

Metódu clustered shading som porovnával s 3 metódami, ktoré sú založené na defered shadingu a to:

1. **Naivný clustered shading** – clustered shading bez fázi vyraďovania svetiel a teda pre každé subfrustum sa prehľadáva list všetkých svetiel v scéne, ktoré ho zasahujú.
2. **Tiled shading** – tiled shading popísaný v [2].
3. **Klasický deferred shading** – pre každý fragment sa počíta osvetlenie priechodom všetkých svetiel v scéne.

Výsledky sú zobrazené na obrázku 8. Z obrázku sa zdá, že metóda clustered shading je len o niečo efektívnejšia ako jej predchodza tiled shading. Je nutné podotknúť, že vzorky sa brali zo scény, kde boli minimálne hĺbkové nesúvislosti a preto sa nedostatky predchádzajúcej metódy neprejavili. Taktiež moja implementácia metódy tiled shading umožňovala len 1024 svetiel pre jednu tile, a preto pri väčšom počte svetiel bolo možné pozorovať artefakty a namerané hodnoty boli skreslené. Zrýchlenie metódy clustered shading voči ostatním spominaním metódam pri 10k svetiel je v tabuľke 1.

Tabuľka 1. Tabuľka znázorňuje faktor zrýchlenia voči testovaným metódam pri 10k svetiel v scéne. Vzorky boli získané zo scény s malými hĺbkovými nesúvyslosťami, a preto zrýchlenie voči tiled shading bolo malé.

Clustered naive	Tiled	Deferred
2.6	1.2	99.2

8. Záver

Clustered shading je technika, ktorou sa snažíme zredukovať počet svetelných kalkulácií a urýchliť tak výpočet osvetlenia. Funguje na princípe delenia viewspace na subfrustá (clustre), ktorým sa priraďuje list potenciálne ovplyvňujúcich svetiel. Oproti predchádzajúcemu tiled-shadingu je robustňejšia v tom, že vytvára menšie subfrustá a teda clustre obsahujú menší počet potenciálne zasahujúcich svetiel.

Kvôli nedostatku času som sa rozhodol pre menej efektívnejšiu metódu radenia založenú na porovnávaní. Je známe, že na masívne paralelných systemoch, ako sú GPU, je radix sort jedna z najlepších metód na zoraďovanie, a preto v budúcej práci by bolo lepšie implementovať ten.

Ďalej by bolo vhodné pridať výpočet tieňov podľa [7], a tak docieliť ešte väčšiu vizuálnu presnosť.

9. Poďakovanie

Táto práca bola vytvorené v rámci bakalárskej práce pod vedením Ing. Tomáša Mileta a týmto by som sa mu chcel poďakovať za cenné rady a skvelé vedenie.

Literatúra

[1] Ola Olsson, Markus Billeter, and Ulf Assarsson. Clustered deferred and forward shading. In *Proceedings of the Fourth ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics*, EGGH-HPG'12, pages 87–96, Goslar Germany, Germany, 2012. Eurographics Association.

[2] Ola Olsson and Ulf Assarsson. Tiled shading-preprint. 2011.

[3] Henry Fuchs, John Poulton, John Eyles, Trey Greer, Jack Goldfeather, David Ellsworth, Steve Molnar, Greg Turk, Brice Tebbs, and Laura Israel. Pixel-planes 5: A heterogeneous multiprocessor graphics system using processor-enhanced memories. *SIGGRAPH Comput. Graph.*, 23(3):79–88, July 1989.

[4] Zina H. Cigolle, Sam Donow, Daniel Evangelakos, Michael Mara, Morgan McGuire, and Quirin Meyer. A survey of efficient representations for independent unit vectors. *Journal of Computer Graphics Techniques (JCGT)*, 3(2):1–30, April 2014.

[5] Aaron Lefohn, Joe M. Kniss, Robert Strzodka, Shubhabrata Sengupta, and John D. Owens. Glift: Generic, efficient, random-access gpu data structures. *ACM Transactions on Graphics*, 25(1):60–99, January 2006.

[6] X. Ye, D. Fan, W. Lin, N. Yuan, and P. Ienne. High performance comparison-based sorting algorithm on many-core gpus. In *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, pages 1–10, April 2010.

[7] Ola Olsson, Markus Billeter, Erik Sintorn, Viktor Kämpe, and Ulf Assarsson. More efficient virtual shadow maps for many lights. *Visualization and Computer Graphics, IEEE Transactions on*, 21(6):701–713, 2015.