

Semestrální projekt předmětu Výpočetní fotograbie – závěrečná zpráva
Tone Mapping: implementace algoritmu Khan20

Milan Tichavský, xticha09@fit.vut.cz

1 Úvod

Zařízení běžně používaná k zobrazování digitálního obsahu, jako jsou monitory, televizory a tiskárny, nejsou schopna zobrazit celý dynamický rozsah světla přítomného v reálném světě. Z tohoto důvodu existují metody pro převod obrazu s vysokým dynamickým rozsahem (HDR) na standardní dynamický rozsah (SDR), aby ho bylo možné správně vizualizovat na dostupných zařízeních. Tento proces se nazývá *tone mapping*.

Cílem této práce je implementace a analýza jednoho z moderních *tone mapping* algoritmů, konkrétně metody Khan20 [1], jako pluginu do Tone Mapping Studio (TMS) [2].

2 Teoretické základy

Pro pochopení *tone mappingu* je nutné zmínit základní vlastnosti lidského vidění. Lidské oko dokáže adaptivně vnímat velký rozsah jasových hodnot díky nelineárnímu vnímání jasu a kontrastu. Toto umožňuje transformovat každou část spektra s jinou přesností a tím lépe využít výstupní rozsah hodnot za zachování co nejvíce detailů.

3 Popis algoritmu Khan20

Algoritmus Khan20 je globální *tone mapping* operátor (TMO), to znamená, že pro transformaci je využita monotónní neklesající mapovací funkce. Algoritmus je založený na transformaci obrazu pomocí funkce *Perceptual Quantizer (PQ)* a histogramu jasu. Schéma algoritmu je vidět na obrázku 1.

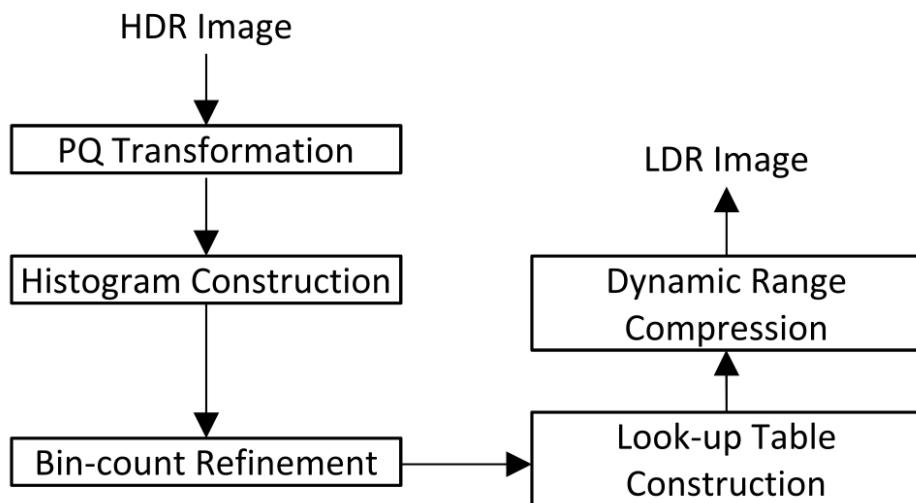
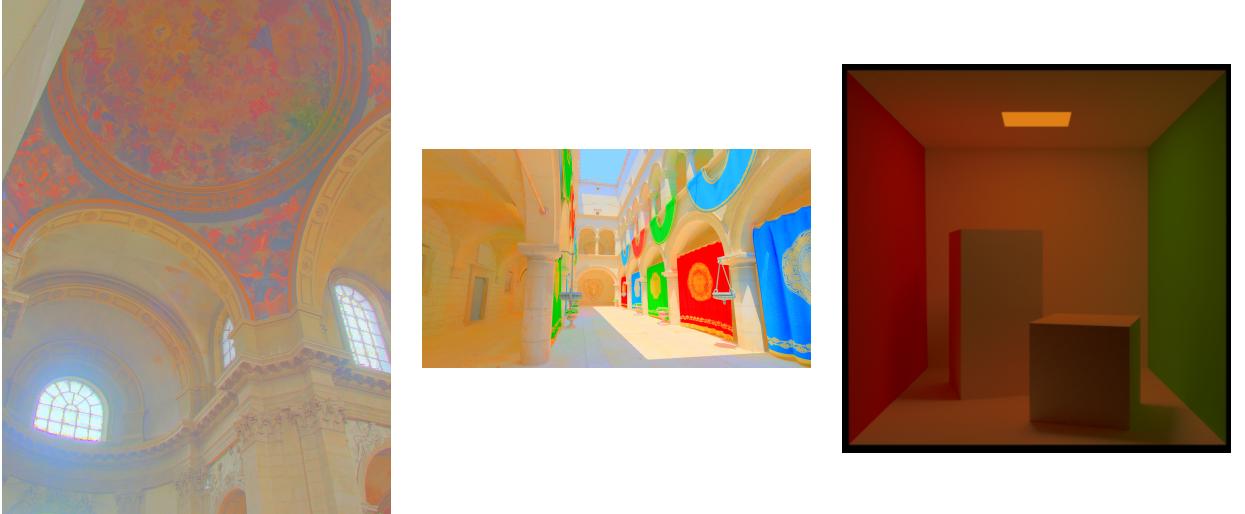


Figure 1: Schéma algoritmu převzato z článku Khan20 [1].

Funkce PQ se snaží simulovat vnímaní lidského oka a transformuje vstupní jas v rozsahu $[0, 10000] \text{ cd/m}^2$ na normalizovaný rozsah $[0, 1]$, přičemž nižší jasové hodnoty mají k dispozici větší relativní rozlišení.

Výstup po transformaci je ukázán v tabulce 1. Je vidět, že z takto transformovaného obrázku lze jasně vidět celou scénu, která ale působí vybledle a má relativně nízký kontrast.

Table 1: Obrázky po transformaci funkcí *Perceptual Quantizer*.



Z toho důvodu je v druhém kroku algoritmu zkonztruován histogram jasů, ze kterého je vytvořena vyhledávací tabulka. Podle této tabulky se poté provádí mapování hodnoty každého vstupního pixelu na hodnotu výstupní. Počet intervalů, na které je histogram rozdělen, je dán parametrem N (v příkazové řádce TMS lze zadat přepínačem `-Bins`), přičemž implicitní hodnota je 256.

Už dříve bylo zjištěno, že použití klasického histogramu je problematické, protože se může stát, že malým oblastem přiřadí moc malou část výsledného rozsahu. Například pokud máme čistě modrou oblohu zabírající velkou část snímku, nechceme, aby zabírala půlku výsledného rozsahu jasu, protože jas většiny pixelů oblohy bude velmi podobný. Proto je nutné omezit počet pixelů v každém intervalu pomocí parametru k , kde $k/N * \text{pixel_count}$ je maximální počet pixelů pro každý interval. S pixelama, které se do daného počtu nevlezou, se nic neděje, akorát pro další výpočty je počet pixelů v daném intervalu podhoden. V TMS lze parametr k nastavit přepínačem `-Truncation` s implicitní hodnotou 5.

3.1 Ukládání výsledného obrázku

Zatímco na vstupu algoritmu se předpokládá RGB obrázek, výstup algoritmu jsou transformované úrovně jasu pro každý pixel. V článku bohužel není uvedeno, jak přesně tento výstup převést do výsledného souboru. Proto jsem zvolil to, co považuji za standardní postup. Tedy obrázek na vstupu jsem převedl do Yxy barevného prostoru, nahradil jasovou složku nově vypočtenými hodnotami a výsledek uložil jako finální obrázek. Pro transformaci do Yxy jsem využil vestavěných metod Tone Mapping Studio.

4 Implementace a výpočetní složitost

Výpočetní složitost implementace je nízká, program projde celkem 4x přes všechny pixely obrázku a v každém kroku provede relativně jednoduché výpočty. Počet průchodů by šel

snížit, ale chtěl jsem nechat kód co nejpřehlednější, takže se každá funkce mapuje na jednu fázi algoritmu.

Z hlediska optimalizací je nejzajímavější implementace vyhledávání v lookup tabulce, pro které jsem použil `std::upper_bound` nad vektorem. Tato funkce sice pracuje s iterátory, což dělá zápis trochu komplikovanější, ale umožňuje nám provádět vyhledávání v LUT s $\log N$ složitostí, což je za mě klíčové, vzhledem k tomu, že se tato operace provádí pro každý pixel obrázku.

5 Výsledky

Table 2: Srovnání různých metod; ve sloupcích zleva doprava jsou Khan20, Drago03 a Ward94. Všechny spouštěny s výchozíma hodnotama parametrů v Tone Mapping Studiu.



V tabulce 2 je vidět porovnání různých metod. Zvolil jsem ty metody, které ve výchozím

nastavení byly schopny rozumně zobrazit všechny tři obrázky.

Jak je vidět, na všech transformovaných obrázcích metody Khan20 jsou scény krásně viditelné. V žádném případě nebylo nutné jakkoliv přenastavovat výchozí parametry, což považuji za jednu z velkých výhod tohoto algoritmu oproti ostatním, které jsem zkoušel, a kdy jsem musel parametry často upravovat.

Z metod v tabulce 2 se mi subjektivně nejvíce líbí Drago03, protože tmavé části jsou opravdu tmavé a barvy mi přijdou přirozenější. Například strop kostela je pro výstup Khan20 v hodně oranžovém odstínu. Na druhou stranu se světlejší barvy hodí pro chodbu druhého testovacího obrázku, kde jsou díky tomu lépe vidět detaily. Líbí se mi i záře kolem okna kostela, která je jednoznačně nejsvítivějším místem celé scény.

6 Závěr

V rámci projektu jsem implementoval plugin Khan20 do Tone Mapping Studio a porovnal jeho algoritmus s jinými metodami. Výsledky ukazují, že s výchozím nastavením dosahuje algoritmus velmi dobrých výsledků napříč různými scénami. Výstupní snímky jsou o něco světlejší a působí méně saturovaně, než bych preferoval z uměleckého hlediska. Na druhou stranu metoda vyniká v zobrazování detailů, a to zejména v tmavých oblastech. Díky tomu jsou snímky lépe viditelné i při odlescích monitoru nebo v silně osvětleném prostředí, takže v ní vidím potenciál pro řadu praktických aplikací.

References

- [1] KHAN, I. R., AZIZ, W., AND SHIM, S.-O. Tone-mapping using perceptual-quantizer and image histogram. *IEEE Access* 8 (2020), 31350–31358.
- [2] ČADÍK, M. Tone mapping studio. GitHub repository, 2025. Accessed: 2025-04-09. Available at: <https://github.com/cadik/TMS>.