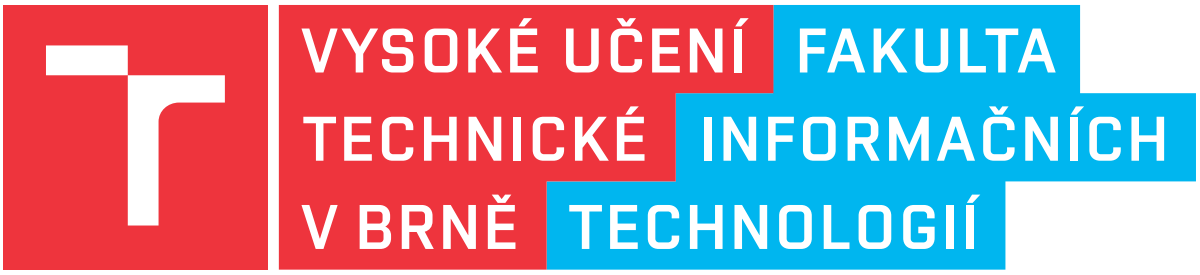


True-color GIF

Multimédia - Projekt

29. apríla 2024

Bc. Peter Ďurica (xduric05)

Obsah

1 Zadanie projektu **1**

2 Obrazový formát GIF **1**
2.1 True-color GIF 1
2.2 Podpora internetových prehliadačov pre true-color GIF 2
2.3 Výsledky prieskumu tvorby true-color GIF 2

3 Tvorba programu **2**
3.1 Bloková metóda 2
3.2 Iteračná metóda 3
3.3 Tvorba animovaného GIFu z videa 4
3.4 Úprava rozlíšenia 5

4 Obsluha programu **5**
4.1 Argumenty 5
4.2 Prerekvizity 6

1 Zadanie projektu

Ako projekt do predmetu Multimédiá som si vybral tému True-color gif. Následne som sa rozhodol, že tento projekt budem riešiť sám. Minimálna povinná časť riešenia bola následovná:

- Zoznámiť sa s obrazovým formátom GIF a zistiť možnosti ako pomocou neho zobraziť true-color obrázky.
- Nájsť prehliadače, ktoré umožňujú korektne zobraziť GIFy so 16 miliónmi farieb.
- Napísať program, ktorý prevedie obrázok s 24 bitovou farebnou hĺbkou do formátu GIF so zachovaním rovnakej farebnej hĺbky.

V zadaní bolo ešte odporúčané rozšíriť túto minimálnu povinnú časť o dobrovolné rozšírenie projektu. Preto som sa rozhodol môj program rozšíriť o schopnosť upraviť rozlíšenie výstupného GIFu, prípadne umožniť tvorbu gifov aj s menej ako 16 miliónmi farieb. Následne mám implementované dva módy na konvertovanie obrázkov do true-color gifu. Ako posledné rozšírenie v programe je schopnosť vytvoriť GIFy z videa, kedy je možné znova určiť rozlíšenie výstupného GIFu, upraviť jeho FPS a nastaviť dithering na jeho snímkoch.

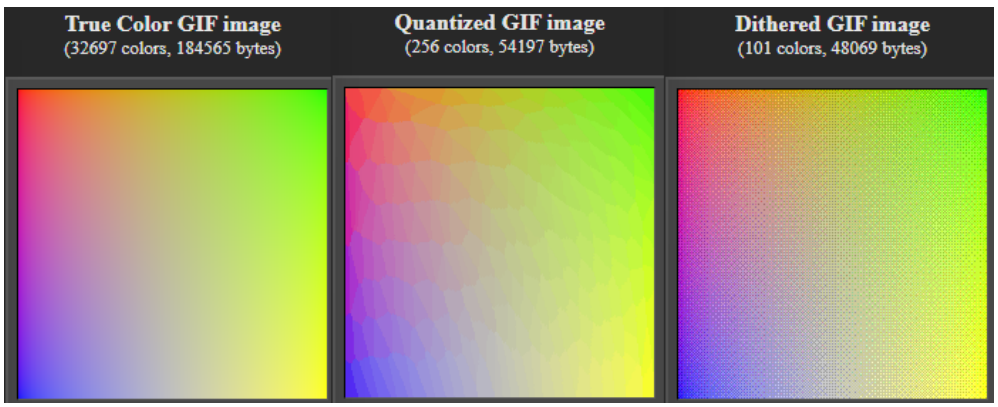
2 Obrazový formát GIF

Táto sekcia sa venuje obrazovému formátu GIF a teórii za možnosťou zobraziť viac ako 256 farieb vo formáte GIF. Následne sa venuje prehliadačom a ich podpore true-color GIF.

2.1 True-color GIF

Chybná predstava [1] o tom, že GIF formát podporuje len 256 farieb vznikla už pri vzniku tohoto formátu v 80. rokoch. Vtedy grafické karty ani viac farieb nepodporovali a preto nebola potreba pre vačší počet farieb. Avšak formát GIF už vtedy podporoval neobmedzený počet farieb. To bolo umožnené špecifikáciou GIF89A, ktorá umožňovala mať neobmedzene veľa obrazových blokov, každý s vlastnou lokálnou paletou farieb. Ďalšou novinkou bolo pridanie transparentnej farby. Takto vieme zakódovať časti obrázku do menších blokov s lokálnymi paletami s 255 farbami plus jednou transparentnou farbou. Keď sa tieto bloky spoja do finálneho obrázku vieme zobraziť viac ako 256 farieb.

Čím viac farieb [1] GIF obsahuje, tým sa logicky zvyšuje aj jeho veľkosť. Keď sa k tomu pridá zlá alebo v niektorých prípadoch aj žiadna kompresia tak sa môže objem týchto gifov veľmi rýchlo rozrásť. Preto ak prehliadač podporuje PNG, true-color GIF by sa nemal použiť.



Obr. 1: Porovnanie [1] True-color GIFu s ostatnými typmi

2.2 Podpora internetových prehliadačov pre true-color GIF

Podľa výskumu z roku 2012 [2] žiadny moderný prehliadač nepodporuje GIFy s nižším oneskorením medzi snímkami ako 20 ms, niektoré dokonca aj viac. Pri osobnej kontrole sa tieto čísla zdajú reálne. Pri GIFoch s nižším oneskorením (0-19 ms) sa toto oneskorenie automaticky nastaví na 100 ms. To znamená že žiadne moderné prehliadače nepodporujú true-color GIFy v originálnej podobe. Tieto GIFy zobrazujú ako normálne animované GIFy s 100 ms oneskorením medzi snímkami. Pravdepodobne je to takto kvôli tomu že je v tejto dobe omnoho výhodnejšie používať PNG alebo iné formáty s plnou 24 bitovou farebnou hĺbkou.

2.3 Výsledky prieskumu tvorby true-color GIF

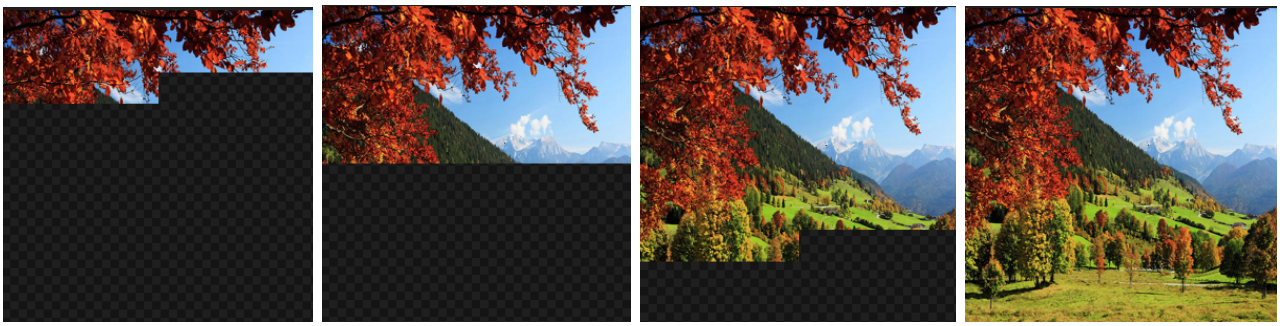
Po zistení si informácií o true-color GIFoch je zrejmé, že tento formát je už v tejto dobe málo využívaný, hlavne kvôli lepším alternatívam a vývoju lepšieho hardvéru. Preto som sa rozhodol že výstupné GIFy z môjho programu budú klasické animované GIFy s oneskorením 20 ms medzi snímkami. K tomuto rozhodnutiu som dospel kvôli tomu, že aj tak moderné prehliadače nemajú podporu pre naozajstné true-color GIFy a takéto riešenie je tak viac vhodné pre modernú dobu a zároveň je aj plynulejšie ako keby som to tlačil do nulového oneskorenia. Takisto mi to umožňuje využiť moderné programovacie jazyky a knižnice, ktoré sú bežne používané pri riešení multimediálnych problémov.

3 Tvorba programu

Pri implementácii som sa rozhodol použiť programovací jazyk Python, na tvorbu GIFov a manipuláciu s obrázkami používam Python knižnicu Pillow. Pri čítaní snímkov z videa využívam knižnicu OpenCV.

3.1 Bloková metóda

Bloková metóda je založená na princípe rozdelenia obrázku na menšie časti (bloky), ktoré by mali teoreticky obsahovať menej farieb ako na celom obrázku. Tým pádom pri kvantizácii palety vieme dosiahnuť omnoho presnejšie farby vzhľadom ku originálnym hodnotám. Kvantizácia je priradenie pixelom na obrázku konkrétnu farbu z obmedzenej palety farieb. Jedným so spôsobom ako sa dajú kvantovať obrázky je algoritmu k-means [3] pre 256 výsledných tried. Avšak knižnica Pillow má vlastné kvantizačné metódy, ktoré dosahujú akceptovateľné výsledky a preto som sa rozhodol ich používať aj v mojom riešení. Jemnou nevýhodou bol však fakt, že metóda na kvantizovanie RGB obrázka je omnoho lepšie implementovaná ako metóda na kvantizovanie RGBA obrázka. Pri riešení sa mi naskytol problém, kedy táto metóda sa samovoľne rozhodla znížiť počet farieb vo výslednej farebnej palete aj keď to nebola nútená robiť.



Obr. 2: Postupné vykreslenie obrázku po blokoch

Rozhodol som sa preto, že budem kvantovať obrázky len z RGB spektra a teda som bol nútený obetovať jednu farbu ako transparentnú. V prípade dostupnosti Alfa hodnôt by mi stačilo použiť ako transparentnú farbu ľubovolnú farbu s alfa hodnotou 0. V mojej implementácii som sa rozhodol že transparentná farba bude úplne biela, teda: 255, 255, 255 v RGB hodnotách. Kvôli tomuto rozhodnutiu zakaždým pri načítaní obrázka mením pixely zafarbené touto bielou farbou na mierne slabšiu, a teda: 255, 255, 254. Vo finále moje oko nevidí medzi týmito farbami rozdiel a nemyslím si že toto rozhodnutie výrazne negatívne ovplyvňuje kvalitu výsledného obrazu.

Pre zachovanie plnej 24 bitovej farebnej hĺbky chceme použiť bloky o veľkosti 16x16 alebo 15x15. Ja som sa rozhodol pre 16x16 pretože to znamená 256 farebných pixelov plus jedna transparentná farba. V tomto prípade teda maximálne 1 pixel v bloku bude mať nesprávnu farbu a to len v nepravdepodobnom prípade ak každý pixel v bloku bude mať vlastný RGB kód farby. Pre presne 255 farebných pixelov sa dajú použiť ešte bloky 255x1 alebo 53x5 ale rozhodol som sa nepoužiť tieto rozmery v mojej implementácii. Jedným z dôvodov bol nedostatočný kvalitatívny rozdiel medzi obrázkami s blokmi s 255 pixelmi a blokmi s 256 pixelmi. Druhý dôvod bol ten, že už implementovaná iteračná metóda dokáže zachovať obrázok zaručene bez straty farebnej informácie.

Samotná bloková metóda prebieha následovne. Užívateľ si zvolí veľkosť jedného bloku. Táto veľkosť môže byť v rozmedzí 16 a viac. V prípade, že veľkosť nieje zadaná alebo je zadaná mimo rozmedzie, je automaticky upravená na 16. Toto číslo znázorňuje jednu stranu štvorcového bloku, ktorý sa bude postupne pridávať do výstupného GIFu.

Keď už poznám rozmery blokov, viem si vypočítať počet stĺpcov a riadkov pre každý blok. Následne si pre každý blok vypočítam jeho pozíciu na obrázku. Tento blok si vytiahnem a vložím ho na nový obrázok, kde všetky pixely mimo vloženého bloku sú zafarbené na bielo (vo finále sú teda transparentné). Takto uložený obrázok prekonvertujem do paletového módu, pričom sa vykoná kvantizácia. Výsledný kvantizovaný obrázok vložím na koniec zoznamu snímkov a pokračujem na ďalší blok.

Pri tvorbe GIFu si nastavím používanie lokálnej farebnej palety pre všetky snímky a zvolím rozostup medzi snímkami na 20 ms bez slučkovania výsledného GIFu. Takýto GIF by mal vykresliť všetky svoje snímky, použiť bielu farbu ako transparentnú a nakoniec zastať s plne vykresleným obrázkom.

3.2 Iteračná metóda

Iteračná metóda sa postupne snaží vykresliť všetky farby na obrázku vo vlnách (iteráciách). Postupne tak paletu po palete a snímok po snímku zostaví finálny obrázok a to vo väčšine prípadov za pomoci menej snímkov a s viac farbami ako bloková metóda, keďže tá nie vždy využije každú farbu palety snímku. Stále však vynechávam úplne bielu farbu, ktorú využívam ako transparentnú. Dôvod prečo toto robím som bližšie opísal v sekcii 3.1.

Priebeh iteračnej metódy je následovný. Najprv si vytvorím slovník, v ktorom má každá farba na na obrázku priradený počet výskytov danej farby a pozície pixelov, ktoré sú ňou zafarbené. Keď už mám zostavený tento slovník viem si zostaviť zoznam, ktorý všetky farby zoradí od najpoužívanejšej po najmenej používanú. Týmto si zaistím, že už v skorých snímkoch by mali byť kontúry výsledného obrázku jasne vyditeľné.

Obr. 3: Postupné iteračné vykreslenie obrazu

Pre každý snímok si postupne z tohoto zoznamu viem zobrať 255 farieb, spoločne z ich pozíciami na obrázku. Bonusová 256. farba bude vždy transparentná biela. Aby som vedel, ktoré farebné pixely môžem preniesť z pôvodného obrázku na nový snímok, musím si vytvoriť masku. Táto maska začína ako pole *False* boolean hodnôt o veľkosti výstupného obrázku. Postupne však tieto hodnoty mením na hodnoty *True* ak sa pozícia pixelu na maske zhoduje s pozíciou pixelu, ktorý je zafarbený ľubovolnou farbou z práve vykresľovanej palety.

Keď je maska vyhotovená, tak pomocou nej nanesiem farby z palety na nový snímok. Pixely s hodnotou *True* zmenia farbu na farbu z obrázku a pixely s hodnotou *False* si nechajú bielu transparentnú farbu. Následne sa finálny snímok uloží do zoznamu snímkov. Pri tvorbe GIFu si zvolím rovnaké nastavenia aké boli opísané v sekcii 3.1 a výsledný GIF uložím.

3.3 Tvorba animovaného GIFu z videa

Pri tvorbe animovaného GIFu si vie užívateľ zvoliť štyri nastavenia. Štartovací čas vo videu, ktorý označuje moment z ktorého sa začne tvoriť GIF. Potom dĺžku GIFu, výstupné FPS GIFu a či chce snímky v GIFe ditherovať. Časy sa vkladajú vo formáte "HH:MM:SS" a automaticky sa prevádzajú na sekundy.

Maximálne podporované výstupné FPS je 30. Ak má video viac FPS tak sa automaticky bude preskakovať taký počet snímkov, aby výsledné video malo menej ako 30 FPS. Jediná implementovaná metóda znižovania FPS je len preskakovanie snímkov, takže hodnota zadaná užívateľom sa zaokrúhli na najbližšiu hodnotu FPS, ktorá je dosiahnuteľná len preskočením určitého počtu snímkov.

Užívateľ má možnosť dvoch módov pri tvorbe GIFu z videa. Prvý mód je predvolený a keď si zvolí, tak sa pre každý snímok videa vytvorí samostatná farebná paleta. Zvolením druhého módu povolí dithering, konkrétne Floyd-Steinbergerov dithering [4]. Knižnica Pillow už obsahuje implementáciu tohto typu ditheringu, ktorú používam pri tomto móde. Teda len stručne opíšem, že pri Floyd-Steinbergerovom ditheringu sa rozdiel medzi reálnou hodnotou pixelu a kvantizovanou hodnotou pixelu pripočíta k hodnotám určitých susedov a tak sa zaistí jemnejší prechod medzi farbami na palete.

Priebeh tvorby animovaného GIFu z videa prebieha následovne. Najprv sa posuniem vo videu na snímok ktorý sa nachádza na zadanom začiatku vytvárania GIFu. Potom si vypočítam koľko snímkov vo videu musím prejsť podľa zadanej dĺžky GIFu a FPS snímaného videa. Následne začnem čítať snímok po snímku obrázky z videa. Ak daný čítaný snímok nepreskakujem tak ho si z neho vytvorím Pillow snímok, ktorý následne prekonvertujem do paletového módu s ditheringom alebo bez podľa nastavenia užívateľa. Výsledný snímok vkladám do zoznamu snímkov.

Výsledný GIF ukladám podobne ako v sekciách 3.1 a 3.2. Rozdiely sú len, že si nastavím rozostup medzi snímkami podľa želaného FPS a nastavím nech sa GIF opakuje donekonečna.

Obr. 4: Rozdiel medzi neditherovaným obrazom (vľavo) a ditherovaným (vpravo)

3.4 Úprava rozlíšenia

Pred prácou na snímku sa upraví jeho rozlíšenie na želané rozmery užívateľa. Maximálne podporované rozlíšenie výstupného GIFu je 1920x1920. Ostatné rozlíšenia sa musia zmestiť tiež do tohto boxu. To znamená že rozlíšenie 1900x2000 nebude podporované, ale napríklad 190x200 bude. Užívateľ takisto môže zvoliť dĺžku len jednej strany snímku, či už šírky alebo výšky. a druhá strana sa dopočíta podľa pomeru strán originálneho obrázka. Toto vypočítané rozlíšenie sa následne ešte porovná s boxom 1920x1920 a ak sa do neho nezmestí tak sa ešte strany upravia. Ak užívateľ nezvolí svoje rozlíšenie, použije sa rozlíšenie obrázka alebo rozlíšenie s rovnakým pomerom ako má obrázok, ale také čo sa zmestí do boxu. Ak si užívateľ zvolí obe strany rozlíšenia obrázku, výsledný GIF nebude brať ohľad na pomer strán pôvodného obrázku a nasilu sa mu upraví rozlíšenie na to zvolené užívateľom. Ak sa rozlíšenie určené užívateľom nezmestí do boxu tak sa upraví aby sa zmestilo.

Všetky tieto úpravy sa vykonajú pred začatím práce na GIFe, aby sa zaistilo, že nebudú nastávať chyby pri spracovaní obrázkov s rozdielnymi rozmermi.

4 Obsluha programu

Program sa spúšťa z príkazového riadku. Nastavenia programu sa určujú argumentami, ktoré sa vložia za názov programu. Toto je pár príkladov ako je možné program spustiť. Hodnoty v `[]` sú nepovinné.

> Video: `python[3] gifcreator.py -v -in video.mp4 -out outputGif[.gif] [-s 1:25:46] -dur 1:10 [-fps 10] [-d] [-w 1280] [-h 720]`
>
> Bloky: `python[3] gifcreator.py -i -in image.png -out outputGif[.gif] [-w 1280] [-h 720] -b [32]`
>
> Iteračne: `python[3] gifcreator.py -i -in image.png -out outputGif[.gif] [-w 1280] [-h 720] [-iter]`

4.1 Argumenty

Argumenty delím na všeobecné, argumenty pre obrázok a argumenty pre video. Zoznam všetkých argumentov a čo robia je následovný. Argumenty v `[]` sú nepovinné. Začnem pri všeobecných argumentoch:

> `(-in alebo -input) inputFilePath` - Cesta ku vstupnému súboru,
>
> `(-out alebo -output) outputFilePath` - Cesta ku výstupnému súboru,
>
> `[(-w alebo -width) wSize]` - Šírka výstupného GIFu (v pixeloch),
>
> `[(-h alebo -height) hSize]` - Výška výstupného GIFu (v pixeloch).

Argumenty pre obrázok sú:

> `(-i alebo -image)` - Nastavenie módu pre obrázok,
>
> `(-b alebo -blocks) [bSize]` - Zapnutie blokovej metódy, prípadne vloženie veľkosti bloku,
>
> `[(-iter)]` - Zapnutie iteračnej metódy (je predvolene nastavená).

A nakoniec argumenty pre video sú:

> `[(-s alebo -start) hh:mm:ss]` - Začiatok snímania GIFu vo videu,
>
> `(-dur alebo -duration) hh:mm:ss` - Dĺžka snímania GIFu,

`[-fps FPS]` - FPS výstupného súboru,

`[-d alebo -dither]` - Zapnutie Floyd-Steinbergovho ditheringu.

Je možné písať viac argumentov rovnakého typu alebo aj priamo argumenty ktoré sa vylučujú. Program vždy vyhodnotí len ten posledný, ktorý je zapísaný.

4.2 Prerekvizity

Program bol vyvíjaný a testovaný na následujúcich verziách knižníc a verzii Python jazyka:

- Pillow 9.5.0,
- NumPy 1.26.4,
- OpenCV 4.8.1,
- Python 3.11.1.

Pri vývoji však neboli použité nejaké extra-moderné metódy alebo funkcie a teda by mal fungovať aj na starších verziách Python 3, prípadne aj Numpy a OpenCV. U Pillow je potrebné mať verziu 8.1.0, kvôli lokálnym paletám pri GIFoch, ktoré boli vtedy pridané.

Zdroje

[1] Howard, P.: *True-Color GIF Example*. [Arch. z 2015-02-22, online], 2000, [cit. 2024-04-28]. Dostupné z: `https://web.archive.org/web/20150222123613/http://phil.ipal.org/tc.html`

[2] Johnson, J.: *Animated GIF Minimum Frame Delay Browser Compatibility Study*. [online], 2012, [cit. 2024-04-28]. Dostupné z: `https://nullsleep.tumblr.com/post/16524517190/animated-gif-minimum-frame-delay-browser`.

[3] Zhao, W.-L.; Deng, C.-H.; Ngo, C.-W.: k-means: A revisit. *Neurocomputing (Amsterdam)*, ročník/vydanie 291, 2018: s. 195–206, ISSN 0925-2312.

[4] Velho, L.; Teixeira, R.: *Floyd-Steinberg Dithering*. [online], 2000, [cit. 2024-04-29]. Dostupné z: `https://www.visgraf.impa.br/Courses/ip00/proj/Dithering1/floyd_steinberg_dithering.html`