# JPEG

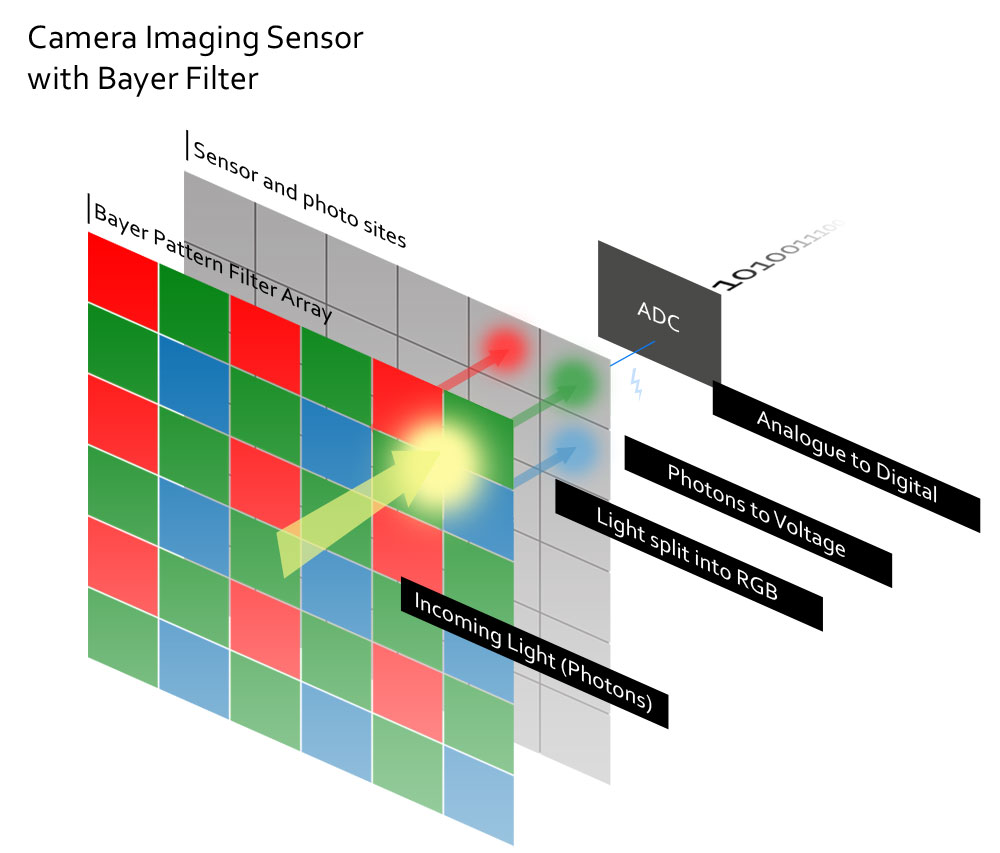
## Povzetek

V tej seminarski nalogi si bomo ogledali delovanje JPEG algoritma. Začeli bomo z motivacijo, kjer si bomo ogledali, zakaj bi sploh želeli uporabljati JPEG algoritem in kaj so njegove funkcije. Nato si bomo ogledali, kako se ustvari slika in nato kako se to sliko obdela z JPEG algoritmom ter kako se jo shrani.

## Motivacija

Slike, ki jih posnamemo s telefonom ali fotoaparatom, so na začetku v formatu RAW, ki je zelo velik (primer slike 1). Mi bi želeli zmanjšati velikost slike tako, da razlike niso opazne, ko gledamo sliko (primer slike 2). Slike sta po velikosti precej različni, vendar se kakovost skoraj ni spremenila. Radi bi analizirali postopek delovanja tega algoritma, saj se ga uporablja povsod, in je dobro vedeti, kako deluje.

## Kako nastane slika.

V kameri imamo senzor, ki je sestavljen iz velikega števila detektorjev. Detektor zazna le intenzivnost svetlobe, ki pride do njega. Zato, kot vidite na sliki, imamo barvni filter, nad posameznim detektorjem prepušča le eno od modre, rdeče, oz. zelene barve.[](https://www.google.com/imgres?imgurl=http%3A%2F%2Fwww.silentpeakphoto.com%2Fwp-content%2Fuploads%2F2020%2F06%2FHow-Image-Sensors-Work.jpg&tbnid=KVe1zUZJj5ZURM&vet=10CBkQMyhzahcKEwiA0KGtiKGFAxUAAAAAHQAAAAAQAw..i&imgrefurl=https%3A%2F%2Fsilentpeakphoto.com%2Fgear%2Fcameras%2Fcamera-guides%2Fhow-do-camera-imaging-sensors-work%2F&docid=EXtTxYTQy0QMKM&w=1000&h=853&q=how%20camera%20works&ved=0CBkQMyhzahcKEwiA0KGtiKGFAxUAAAAAHQAAAAAQAw)

Slika 1 senzor v kameri

Opazimo lahko, da je zelene barve veliko več kot modre in rdeče. To je zato, ker ima naše oko veliko več receptorjev za zeleno kot modro in rdečo barvo, zato je za naš vid veliko bolj pomembna zelena barva. Torej, en kvadratek 2x2 predstavlja en piksel. Našo sliko sedaj lahko direktno shranimo in temu rečemo format RAW. Lahko pa sliko malo obdelamo s postopkom demosaiciranja in dobimo že znan RGB format. Tu je potrebno opozoriti, da za ta postopek obstaja veliko različic, ki so zapletene. Mi se na to obdelavo slike ne bomo preveč ozirali, temveč bomo predpostavili, da dobimo sliko že podano v RGB formatu.

## JPEG algoritem

Naši vhodni podatki so torej matrika RGB. To je matrika, ki ima na vsaki komponenti oziroma pikslu zapis za intenziteto rdeče, zelene in modre barve.

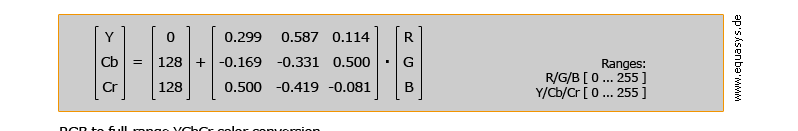
### Obdelava barve

Naše oči veliko bolje zaznavajo spremembe svetlosti kot spremembe v barvah. Ravno zaradi tega pojava bomo pretvorili barve v obliki Y, Cr in Cb, kjer Y predstavlja svetlost, Cr in Cb pa kromatičnost.

Slika, ki vsebuje besede posnetek zaslona, besedilo, barvitost

Opis je samodejno ustvarjen

Slika 2 kako izgleda kromatičnost

To pretvorbo naredimo preko formule. Obstaja veliko različnih enačb, ena lahko vidimo tukaj:

Slika 3 formule RGB v YCrCb

[<https://stackoverflow.com/questions/35595215/conversion-formula-from-rgb-to-ycbcr>]

Sedaj imamo Y, Cr in Cb komponente. Kot smo že omenili, je naše oko veliko bolj občutljivo na komponento svetlosti in veliko manj na kromatične komponente. Zato lahko že zdaj zmanjšamo sliko tako, da kromatične komponente zmanjšamo na četrtino. To bomo naredili tako, da za vse 4x4 blokce, ki se ne sekajo, kromatične komponente povprečimo, in vrednost shranimo v novo matriko. Tako dobimo dve matriki, eno za Cr, drugo za Cb, ki sta sedaj četrtina osnovne velikosti slike.

### Kosanje slike

Pri nadaljnji obdelavi slike se bomo osredotočili le na komponento svetlosti. Moramo se zavedati, da obdelamo tudi ostali dve komponenti na skoraj isti način, zato bomo samo omenili, ko so razlike.

Najprej moramo torej razkosati matriko komponente na 8x8 bloke. To naredimo, da jih lažje obdelujemo. Velika večina kamer v sodobnem času že poda dimenzije slik, ki so deljive z 8, torej to ni problem. Vendar včasih pride do problema. Takrat imamo situacijo:

Slika, ki vsebuje besede posnetek zaslona, črno in belo, narava, voda

Opis je samodejno ustvarjen

Slika 4 Kosanje slike

Kot opazimo na sliki obstaja veliko različnih postopkov, kako sliko dopolnimo, da so dimenzije deljive z 8.

### Diskretna kosinusna transformacija (DCT)

Kot vhodne podatke za DCT algoritme dobimo matriko dimenzije 8x8, saj so naši bloki, ki smo jih razkosali, dimenzije 8x8. Torej, za vsak 8x8 blokec imamo 64 komponent svetlosti (Y). Poudariti je treba, da za DCT algoritme posebej vnesemo Y, Cr in Cb komponente, in ne vse naenkrat.

Kot izhodni podatek dobimo novo 8x8 matriko, kjer vrednost v matriki predstavlja, kako pogosto se pojavi določen izmet teh baznih vzorcev:

Slika, ki vsebuje besede vzorec, posnetek zaslona, tkanina, rdeča

Opis je samodejno ustvarjen

Slika 5 Bazne slikice DCT

Za globje razumevanje diskretne kosinusne transformacije predlagam, da si ogledate video [<https://www.youtube.com/watch?v=Q2aEzeMDHMA> in <https://www.youtube.com/watch?v=lQnUex31f5Q>]. V teh videih je lepo razloženo, kako sama transformacija deluje.

Na kratko, če vas ne zanima ozadje delovanja, si lahko ogledamo le formulo:

Slika, ki vsebuje besede besedilo, pisava, bela, vrstica

Opis je samodejno ustvarjen

Slika 6 Formula za izračun DCT

V formuli N predstavlja dimenzijo našega bloka, v našem primeru je N enak 8. Pixel (x, y) predstavlja vrednost v naši 8x8 matriki, ki je omejena med 0 in 256, kot so omejene vrednosti za Y, Cr in Cb. DCT (i, j) predstavlja i, j-to komponento v naši izhodni matriki.

Pomembno je poudariti, da poznamo tudi inverz naše DCT, ki ga bomo potrebovali kasneje.

### Kvantizacija

Kvantizacija je glavni postopek, pri katerem zmanjšujemo velikost in hkrati kakovost slike.

Za kvantizacijo moramo vpeljati pojem kvantizacijske matrike. Kvantizacijska matrika je matrika velikosti 8x8, v našem primeru.

Slika, ki vsebuje besede besedilo, pisava, posnetek zaslona, bela

Opis je samodejno ustvarjen

Slika 7 Primer kvantizacijske matrike

Kvantizacijska matrika je tesno povezana z matriko, ki jo dobimo na koncu DCT algoritma. Naše oči so veliko bolj občutljive za razlike v blokih z nizko spremembo frekvenc.

Slika, ki vsebuje besede vzorec, posnetek zaslona, tkanina, rdeča

Opis je samodejno ustvarjen

Slika 8 Bazne slikice pri DCT

Na primeru te slike lahko to tudi izkusimo; sličice, ki so levo zgoraj, izgledajo veliko bolj raznolike kot sličice desno spodaj. Ravno zaradi tega pojava uvedemo našo kvantizacijsko matriko, ki ima večje vrednosti desno spodaj ter manjše vrednosti levo zgoraj. Obstaja veliko različnih kvantizacijskih matrik (lahko generiramo tudi svoje). Kot smo omenili že prej, je naše oko veliko bolj občutljivo za svetlobo kot na kromatičnost, to znanje bomo uporabili tudi sedaj in to tako, da bomo za kromatičnosti in svetlost zgenerirali različne kvantizacijske matrike.

Slika, ki vsebuje besede besedilo, posnetek zaslona, pisava, številka

Opis je samodejno ustvarjen

Slika 9 Kvantizacijekse matrike

Da sedaj izvedemo proces kvantizacije delimo istoležne elemente v naši matriki, ki smo jo dobili po DCT postopku ter nači kvatizacijski matiki. Zdeljena števila sedaj tudi zaokrožimo.

Na tak način dobimo zelo veliko ničel v naši matriki, kar nam zelo olajša shvarnjevanje slike in posledično zmanjša velikost datoteke.

Slika, ki vsebuje besede vhodna naprava, periferen, kvadrat, posnetek zaslona

Opis je samodejno ustvarjen

Slika 10 Primer uporabe kvantizacijske matrike na matriki po postopku DCT

Lahko si tudi ogledate video : <https://www.youtube.com/watch?v=aFbGqXFT0Nw> [2:15,5:00]

### Linearna transformacija

Sedaj smo na točki ko si moramo le zapomniti naše informacije na pameten način. Problem nastane, saj v računalnik ne moremo shraniti direktno matrike velikosti 8x8 saj je spomin v računalniku linearen. Zato moramo naše matrike pridobljene po kvantizaciji obdelati na pameten način in na njih izvesti nekakšno linearno transformacijo.

Oglejmo si našo matrikao po kvantizaciji.

Slika, ki vsebuje besede posnetek zaslona, besedilo

Opis je samodejno ustvarjen

Slika 11Matrika po kvantizaciji

Opazimo da je v zgornjem levem kotu zelo velika številka. Če bi pogledali ostale matrike po kvantizaciji bi opazili dve stvari:

1. V zgornjem levem kotu je zmeraj zelo velika številka
2. Matrike od 8x8 blokov, ki so si blizu imajo vrednost v zgornjem levem kotu podobno(absolutna razlika je majhna)

Zaradi teh dveh dejstev bomo to zgornjo levo komponento, ki jo včasih imenujemo tudi DC (ostale komponente imeujemo AC) obdelali na poseben način.

#### Obdelava DC-ja

Iz celotne slike vzemimo vse DC komponente ter jih postavimo v matriko.

Vemo, da zapis veliko števil porabi več bitov informacije kot tapis malih števil. Zato bomo naša števila v matriki DC zmanjšali na naslednji način.

Kjer DC0 predstavlja prvi stolpec matrike DC in d0 ki predstavlja prvi stoplec spremenjene matrike.

Slika, ki vsebuje besede besedilo, rokopis, vrstica, grafični prikaz

Opis je samodejno ustvarjen

Slika 12 Primer predelave DC matrike

#### Zig-Zag sprehod

Najlažje si bo ogledati delovanje zig-zag postopka na sliki:

A diagram of a square with arrows and lines

Description automatically generated

Slika 13 zig zag sprehod

Če naš zig zag sprehod sedaj izvedemo na naši matriki po kvantizaciji, bo izgledal tako:

Slika, ki vsebuje besede besedilo, posnetek zaslona, številka, pisava

Opis je samodejno ustvarjen

Opazimo, da po 14stem elementu nastopijo same ničle. Ravno za to, da imamo na kocu večino ničel uporabimo zig zag sprehod, saj je dokazano, da bo na tak način zelo veliko ničel na koncu našega seznama.

Element obarvan z rdečo barvo je DC komponenta. To DC komponento zamenjamo z elementom iz matrike **d** na **sliki 12.**

### Kodiranje po dolžini(RLE)

Po zig zag sprehodu imamo torej naš 8x8 matriko predstvajleno v linearni obliki. Sedaj da skrajšamo dolžino zapisa uvedemo nov način zapisovanja.

V tem novem načinu so elementi dvojice kjer prvi člen predstavlja ponovitev elementa ki je drugi(drugi člen)

Oglejmo si to na 2h primerih:

* {76, - 11, - 18, 0,1, -3, -2, -1, 1, 1, 0, 0, -1, 1, 0,...., 0}
* (1, -11), (1, - 18), (1, 0), (1, -3), (1, -2), (1, -1), (2, 1), (2, 0), (1, -1), (1, 1), (50, 0)
* {100, 5, -3, 5, 0, 0, 1, 8, -3, 2, -1, -1, -1, 0, 0, 0, 1, 0,..., 0}
* (1, 5), (1, -3), (1, 5), (0, 2), (1, 1), (1, 8), (1,-3), (1, 2), (3, -1), (3, 0), (1, 1), (47,0)

Opomba: različni JPEG algoritmi opravijo kodiranje po dolžini na razične načine.

### Huffmanovo kodiranje

Delovanje huffmanovega kodiranja si oglejte na posnetku:

<https://www.youtube.com/watch?v=iEm1NRyEe5c>

V primeru, da vas zanima več si oglejte: <https://www.youtube.com/watch?v=B3y0RsVCyrw>

V obeh primerih zgeneriramo Huffmanovo drevo za primere črk. Sedaj bomo to prilagodili tako da bomo namesto črk uporabili dvojice kot so (1,5), (50,0),… te dvojice predstavljajo komponente v našem kodiranju po dolžini. Na tak način precej zmanjšamo količino bitov ki jih potrebujemo za shranjevanje naših informacij. Podobno idejo uporabimo tudi ko kodiramo DC komponente naše slike.

Tu lahko sedaj povemo, da nekateri JPEG algoritmi uporabljajo že standardizirana Huffmanova drevesa ter tako prihraniš še več prostora. V drugih algoritmih pa lahko mi zgeneriramo svoja Huffmanova drevesa in imamo na tak način bolj optimiziran zapis vendar moramo hkrati po pošiljamo informacije o sliki poslati tudi Huffmanovo drevo, kar seveda porabi več prostora.

### Zapis in shranjevanje podatkov

<https://www.ccoderun.ca/programming/2017-01-31_jpeg/>