UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Peter Kragelj

ZAJEMANJE SLIK VISOKEGA DINAMIČNEGA RAZPONA Z VEČKRATNO IZPOSTAVITVIJO

Diplomska naloga
na univerzitetnem študiju

Mentor: prof. dr. Aleš Leonardis

Ljubljana, 2008

Rezultati diplomskega dela so intelektualna lastnina Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavljanje ali izkoriščanje rezultatov diplomskega dela je potrebno pisno soglasje Fakultete za računalništvo in informatiko ter mentorja.

Zahvala

Na prvem mestu bi se rad zahvalil svojemu mentorju, prof. dr. Alešu Leonardisu, ki mi je svetoval pri pripravi tega diplomskega dela. Nato bi se rad zahvalil svojemu dekletu Veroniki in vsem domačim, ki so mi vedno stali ob strani. Nazadnje pa bi rad izrekel zahvalo tudi vsem prijateljem in sodelavcem, ki so kakorkoli pripomogli pri izdelavi tega diplomskega dela.

Za Veroniko ...

Kazalo

Seznam uporabljenih kratic in simbolov

- CIE - *Commission internationale de l'éclairage*, mednarodna komisija za svetlobo
- CMYK - barvni prostor s subtraktivnim modelom mešanja barv, poimenovan po angleških osnovnih barvah *(C - cyan, M - magenta, Y - yellow in K - key (črna))*
- EXR - *extended range*, format VDR-slik
- HDR - *high dynamic range*, visok dinamični razpon, tudi ime formata slik s takšnim razponom
- LCD - *liquid crystal display*, vrsta prikazovalnika, ki uporablja tekoče kristale
- NDR - nizek dinamični razpon
- PPM - *portable pixmap*, format slik s preprosto strukturo
- RGB - barvni prostor, ki uporablja additvini barvni model, poimenovan je po angleških besedah osnovnih treh barvah *(R - red, G - green, B - blue)*
- TIFF - *Tagged Image File Format*, format slik, ki lahko vsebuje tudi VDR-slike
- VDR - visok dinamični razpon
- $V(\lambda)$ - funkcija spektralne občutljivosti človeškega očesa
- XYZ - barvni prostor definiran s strani komisije CIE

- YC_bC_r - barvni prostor, kjer je svetlost Y ločena od barvnih kanalov.

Povzetek

V svetu je vse več aplikacij, katerih sestavni del je tudi digitalna fotografija. Slike so zajete z modernimi aparati vendar velika večina teh slik uporablja povsem enak format shranjevanja.

Za prikaz vsakega barvnega kanala format uporablja 8 bitov, to pomeni, da imamo na voljo le 256 vrednosti za prikaz celotnega barvnega razpona. To pa je premalo za prikaz veliko prizorov, posebno takšnih, ki vsebujejo tako temne kot tudi zelo svetle dele slike.

Zato se v svetu razvija nova vrsta slik, slike visokega dinamičnega razpona. Te slike zmorejo prikazati veliko višji dinamični razpon, zato ne prihaja do problemov s pre- ali podosvetljnimi deli slik.

Slike visokega dinamičnega razpona je potrebno najprej zajeti, to pa lahko naredimo na več načinov. Z različnimi naprednimi tehnologijami, ki uporabljajo posebne senzorje, pa tudi s povsem programsko rešitvijo z večkratnim zajemanjem pri različnih časih izpostavitve.

Zajemanje slik visokega dinamičnega razpona z večkratno izpostavitvijo sestavlja več algoritmov, ki se med seboj dopolnjujejo in skupaj sestavijo postopek, ki iz več slik nizkega dinamičnega razpona istega prizora generira eno sliko z visokim dinamičnim razponom. Slike je potrebno najprej medsebojno poravnati, izračunati je potrebno funkcijo odziva senzorja, nato slike združiti, na koncu pa še odstraniti morebitne napake, ki nastanejo.

Ključne besede:

visoki dinamični razpon, fotografija, zajemanje, izpostavitev ...

Poglavje 1

Uvod

1.1 Slike visokega dinamičnega razpona

V svetu je vse več aplikacij, katerih sestavni del je tudi digitalna fotografija. Slike so zajete z modernimi aparati in skenerji, generirane z naprednimi grafičnimi tehnikami ali pa zgenerirane z različnimi programi za risanje. Velika večina teh slik uporablja povsem enak format shranjevanja.

V svojem življenjskem ciklu slike preidejo skozi več transformacij. Najprej so ustvarjene z eno izmed poprej naštetih tehnik. Nato jih je potrebno shraniti na digitalni medij. Najverjetneje jih še dodatno obdelamo z želenimi grafičnimi tehnikami, nazadnje pa jih prikažemo na prikazovalniku ali pa jih natisnemo.

Trenutni trend stremi k zajemanju slik z visoko ločljivostjo. Uporabniški fotoaparati že podpirajo ločljivosti nad 5 mpik, medtem ko so na voljo že senzorji z več kot 12 mpik. Nič ne kaže da se bo ta trend v kratkem ustavil. Seveda je za dober prikaz ločljivost pomembna, kar prikazuje slika 1.1, vendar smo pred velikim premikom v razmišljanju v digitalni fotografiji.

Najlažje je problem dinamičnega razpona obrazložiti na sivinski sliki. Trenutno se uporablja za prikaz vsakega slikovnega elementa slike 8 bitov, to pomeni, da imamo na voljo le 256 vrednosti za prikaz kateregakoli slikovnega elementa na sliki. Tudi pri barvni fotografiji je problem enak. Ravno tako se uporablja 8 bitov na slikovni element, vendar 8 bitov za predstavitev vsakega izmed barvnih (rdeča, zelena, modra) kanalov. S tremi zlogi na točko lahko prikažemo več kot 1.6 milijona barv. Na prvi pogled je to sicer zelo impresivna številka, vendar imamo še vedno na voljo le 256 vrednosti za predstavitev posameznega barvnega kanala. To pa je premalo za prikaz veliko prizorov. Najbolj razširjen primer je slika notranjosti, na kateri se nahaja okno (slika 1.2, levo). Takšna postavitev pripelje do temnih kot tudi zelo svetlih delov slike. Ta-

Slika 1.1: Pomembnost ločljivosti v fotografiji. Na levi slika z zmanjšano ločljivostjo 75 x 100 slikovnih elementov, na desni pa original s 600 x 800 slikovnimi elementi.

kšnega razpona pa slika z 8-bitnimi barvnimi kanali ni sposobna učinkovito prikazati [10].

Na sliki 1.2 desno je enak prizor pripravljen z metodami visokega dinamičnega razpona - VDR. Preden je bila natisnjena, jo je bilo zaradi omejitev tiskalnika potrebno pretvoriti v nizek dinamični razpon - NDR, ki je lahko natisnjen. Vidimo lahko, da je osvetlitev tako na temnih kot tudi na svetlih delih izboljšana. Ugotovimo lahko tudi, da je visoko dinamičen razpon uporaben, tudi kadar nimamo na voljo medija, ki bi bil sposoben prikaza takšnega razpona. S takšnim medijem pa bi bile razlike v kakovosti še veliko večje.

Pojavlja se veliko vprašanj pri uporabi slik VDR: Kako sliko zajeti, kakšen dinamičen razpon je sploh smiselno zajeti, kakšne algoritme pri tem uporabiti, kako jo shraniti (format, medij), kako jo prikazati (različni problemi pri različnih tehnologijah)?

1.2 Zajemanje slik visokega dinamičnega razpona

Še posebno zanimivo področje je zajemanje slik visokega dinamičnega razpona, ki omogoča več različnih pristopov k reševanju problema.

Dinamični razpon, ki se sedaj uporablja za zajemanje slik, je približno dva

Slika 1.2: Primer prizora z velikim dinamičnim razponom. Levo NDR-slika, desno VDR-slika [10].

velikostna reda, shranjen kot zlog, za vsakega izmed barvnih kanalov. Veliko večjega razpona tudi ni mogoče prikazati natisnjenega na papir, zato bomo za prikaz izboljšanja uporabili zmanjšane vrednosti. Slika 1.3 levo tako prikazuje sliko z 8 biti na barvni kanal, medtem ko je na desni enak prizor s samo 4 biti na barvni kanal. Opazimo lahko, da manj bitov pomeni tudi slabšo kvaliteto.

Tabela 1.1: Svetlost okolja za značilne vire svetlobe

Okolje	**Svetlost** $[cd/m^2]$
Zvezdna svetloba	10^{-3}
Mesečina	10^{-1}
Notranjost	10^{2}
Sonce	10^{5}
Maksimalna intenziteta navadnega monitorja	10^{2}

Osem bitov, kot jih uporabljajo današnje fotografije, je dovolj za marsikateri prizor, vendar je veliko prizorov, ko je to premalo. Eden izmed razlogov je, da resnični svet ustvarja razpone, ki so veliko večji od dveh velikostnih redov. Na primer sonce ob poldne je tudi 100 milijonkrat svetlejše kot svetloba zvezd

Slika 1.3: Razlika v kakovosti slike zaradi majšega dinamičnega razpona. Levo 8 bitov na barvni kanal, desno 4 biti na barvni kanal [14].

ponoči. Tipična svetlost v različnih okoljih je podana v tabeli 1.1.

Človeško oko je sposobno prilagoditve svetlobnim pogojem, različnim za skoraj 10 velikostnih redov. Znotraj enega prizora pa je sposobno zaznati razpon svetlosti petih velikostnih redov [5].

1.2.1 Tehnike reševanja problema visokega dinamičnega razpona

Ker je človeško oko sposobno zaznati veliko večji dinamični razpon, kot ga lahko zajamejo današnji fotoaparati, so se razvile različne tehnike, ki se trudijo izboljšati dinamični razpon zajetih slik.

Nekatere, so popolnoma programske tehnike z uporabo večkratnih izpostavitev, kjer je prednost predvsem ta, da lahko uporabimo tehnologijo, ki je že sedaj na voljo. Sam dinamičen razpon pa razširimo z uporabo več slik, vendar pri tem nastajajo problemi zaradi dolgega trajanja zajemanja slik. Med posameznimi zajetji se prizor lahko spremeni, objekti na njem se premaknejo in tako nastanejo napake zaradi gibanja, tako imenovani "duhovi". Te napake je zato potrebno reševati z bolj zapletenimi algoritmi, ki jih čim bolj učinkovito odstranijo.

Obstaja tudi vedno več rešitev z novimi, modificiranimi senzorji, ali pa več senzorji znotraj fotoaparata, ki na različne načine povečujejo dinamični razpon. Takšni so senzorji z različno občutljivimi senzornimi elementi, z več senzornimi elementi za en slikovni element ali pa z visokotehnološkimi senzorji, ki so sami po sebi sposobni zajeti večji dinamični razpon. Pri vseh teh novih tehnologijah je problem predvsem cena in dobavljivost, saj tehnologija še ni zrela za uporabniške aplikacije.

1.2.2 Uporabnost slik visokega dinamičnega razpona

Čeprav današnji prikazovalniki in tiskalniki niso sposobni prikazati visokega dinamičnega razpona, to še ne pomeni, da so slike visokega dinamičnega razpona neuporabne. Trenutno so slike kodirane z 8-bitnimi barvnimi kanali, kar se navadno opravi že pri zajetju. Tako že takrat za vedno izgubimo veliko informacij, ki jih je vseboval dani prizor. To pa ni optimalno. Najbolje bi seveda bilo, da bi slike zajeli vsaj s tolikšnim dinamičnim razponom, kolikor ga je sposobno zaznati človeško oko. Strmimo pa k temu, da je dinamični razpon čim večji. Takšne slike je namreč vedno mogoče pretvoriti v slike nizkega dinamičnega razpona, ki jih je sposobna prikazati današnja tehnologija, pri tem pa imamo na voljo več informacij in s tem bolj kakovostne rezultate.

Slike visokega dinamičnega razpona omogočajo tudi lažjo nadaljnjo obdelavo, kajti ne vsebujejo preosvetljenih slikovnih elementov [10].

Nenazadnje je potrebno omeniti tudi napredek na področju monitorjev, kajti že nekaj let obstajajo prototipi monitorjev visokega dinamičnega razpona. Ko bodo le-ti prešli v splošno uporabo, bodo slike visokega dinamičnega razpona postale standard, kajti na teh monitorjih bodo navadne slike nizkega dinamičnega razpona enake, medtem ko bodo slike visokega dinamičnega razpona zasijale v vsej svoji kvaliteti [13].

1.3 Vsebina diplomskega dela

V tem diplomskem delu je podan pregled področja slik visokega dinamičnega razpona, še posebno pa je razdelan problem zajemanja tovstnih slik. Namen tega dela je namreč preučiti področje zajemanja slik visokega dinamičnega razpona, implementirati izbrane tehnike zajemanja in jih praktično preizkusiti.

V drugem poglavju so podane bistvene teoretične osnove slik visokega dinamičnega razpona in drugi problemi ter rešitve, ki se pojavljajo. V tretjem poglavju so opisani različni načini zajemanja slik visokega dinamičnega razpona, medtem ko četrto poglavje opisuje zajemanje VDR-slik z večkratno izpostavitvijo. Peto poglavje vsebuje opis implementacij teh tehnik v tem delu, šesto pa rezultate njihovega preizkusa. Sedmo poglavje podaja sklepne ugotovitve. V prilogah se nahaja izvorna koda narejene implementacije.

Poglavje 2

Teoretične osnove

2.1 Svetloba in barve

Za popolno razumevanje napredka, ki ga prinašajo slike z visokim dinamičnim razponom, si je najprej potrebno ogledati nekatere osnove s področja fotografije. VDR-slike so namreč povezane z obstoječimi disciplinami, kot so radiometrija, fotometrija in izgled barv. Vsaka od teh disciplin se ukvarja s svojim pogledom na svetlobo in njeno zaznavanje.

2.1.1 Radiometrija

Izraz prizor se nanaša na naravno ali umetno okolje, ki lahko postane tema slike. Takšno okolje vsebuje predmete, ki odbijajo svetlobo.

Tabela 2.1: Radiometrične količine

Količina	**Enota**	**Definicija**
Energija sevanja	J (joule)	Q_e
Moč sevanja	$Js^{-1} = W$ (watt)	$P_e = \frac{dQ_e}{dt}$
Izsevanost	Wm^{-2}	$M_e = \frac{dP_e}{dA_e}$
Obsevanost	Wm^{-2}	$E_e = \frac{dP_e}{dA_e}$
Jakost sevanja	Wsr^{-1}	$I_e = \frac{dP_e}{d\omega}$
Sevalnost	$Wm^{-2}sr^{-1}$	$L_e = \frac{d^2P_e}{dA\cos\theta d\omega}$

Radiometrija je znanost, ki se ukvarja z merjenjem svetlobe. Svetloba je sevana energija, izražena v Joulih. Ker se svetloba giblje skozi prostor, zrak

in vodo, nas zanimajo predvsem izpeljane količine, ki merijo, kako se svetloba širi. To vsebuje predvsem sevano energijo, merjeno preko časa, prostora ali kota. Definicije najpomembnejših količin so podane v Tabeli 1.2.

Vir svetlobe s sevanjem oddaja energijo in moč. Govorimo o sevalnem toku ali fluksu, ki predstavlja energijo, ki jo vir izseva v enoti časa. Obsevanost in izsevanost sta merili za količino sevalnega toka, ki pada na neko ploskev oziroma se izseva z določene ploskve. Ker vir ne seva energije v vse smeri enakomerno, lahko govorimo o jakosti sevanja, ki je definirana s pomočjo sevalnega toka v enoti prostorskega kota. S sevalnostjo pa označimo jakost sevanja določene ploskve pod določenim kotom [10].

2.1.2 Fotometrija

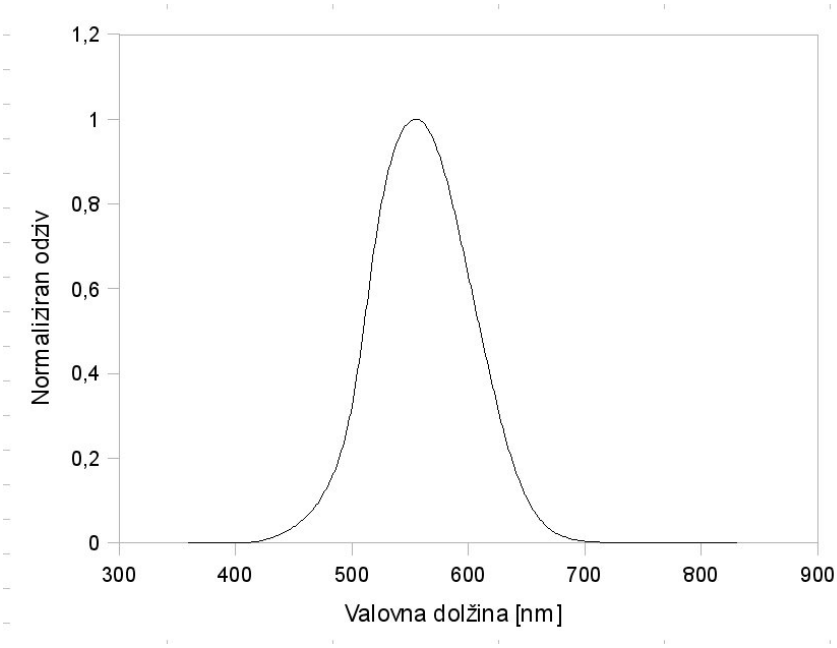
Fotometrija je veja radiometrije, torej vede, ki se ukvarja z merjenjem elektromagnetnega sevanja. Fotometrija se ukvarja samo z vidnim delom spektra elektromagnetnega sevanja, ki sega od 380 nm pa do 830 nm valovne dolžine. Ta del spektra imenujemo vidna svetloba.

Vsaka radiometrična količina ima v fotometriji svojega dvojnika (Tabela 1.3), od katerega se razlikuje po tem, da upošteva spektralno občutljivost človeškega vida

Tabela 2.2: Fotometrične količine

Količina	**Znak**	**Enota**
Svetlobna moč	P	lm (lumen)
Svetlobna energija	Q_v	$lm\,s$
Svetlost vira	M_v	$lm\,m^{-2}) = lx$ (lux)
Osvetljenost	E_v	$lm\,m^{-2}) = lx$ (lux)
Svetilnost	I_v	$lm\,sr^{-1} = cd$ (kandela)
Svetlost	L_v	$cd\,m^{-2}$

Oko namreč ni enako občutljivo skozi celoten razpon. Občutljivost se razlikuje tudi med posamezniki, vendar so ta odstopanja majhna, tako da jo lahko podamo z eno samo krivuljo $V(\lambda)$ (slika 2.1).



Slika 2.1: Odziv standardnega opazovalca na svetlobo [12].

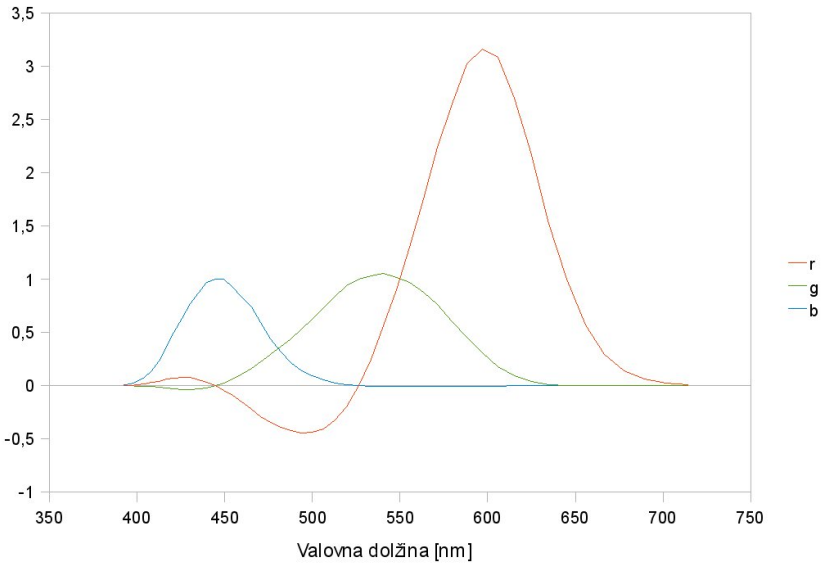
2.1.3 Merjenje barv

S pomočjo eksperimentov enačenja barv so raziskovalci uspeli določiti krivuljo zaznavanja posameznih barv (slika 2.2). Eksperiment poteka tako, da je monitor razdeljen na polovico. Ne eni strani se nahaja stimulus, drugo stran pa udeleženec eksperimenta spreminja z nastavljanjem intenzitete treh osnovnih barv, dokler se polovici ne ujemata. Vsaka barva je tako izražena s formulo

$$Q_\lambda = \bar{r}(\lambda)R + \bar{g}(\lambda)G + \bar{b}(\lambda)B, \tag{2.1}$$

kjer $\bar{r}(\lambda)$, $\bar{g}(\lambda)$ in $\bar{b}(\lambda)$ predstavljajo vrednosti barvnih filtrov, R, G, B pa prilegajoče se uteži. Človeško oko ima namreč receptorje za kratke, srednje in dolge valovne dolžine svetlobe, zato lahko vsako barvo definiramo kot trojček intenzitet osnovnih barv.

Leta 1931 ustanovljena mednarodna komisija za svetlobo (Commission internationale de l'éclairage), kratko CIE, je definirala standard, kjer so vse barve izražene samo s pozitivnimi vrednostmi (slika 2.3). Vrednosti za $Y(\lambda)$ so namenoma izbrane tako, da sovpadajo z $V(\lambda)$, tako torej $Y(\lambda)$ predstavlja fotometrično uravnoteženo količino [10].



Slika 2.2: Rezultati eksperimenta enačenja barv [12].

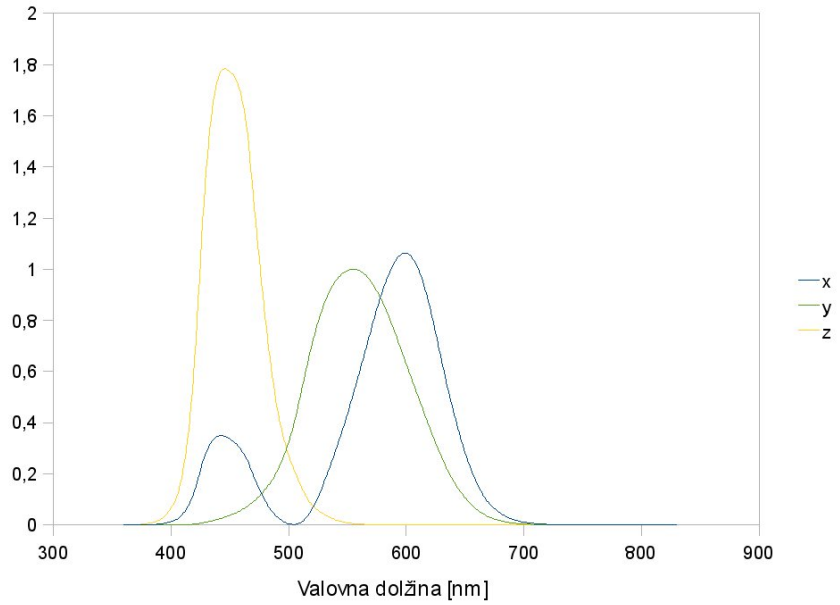
2.1.4 Barvni prostor

Barvni prostor predstavlja množico vseh barv znotraj barvnega obsega tega prostora. Ponavadi je izrazen z matriko, ki prikazuje preslikavo iz CIE XYZ-barvnega prostora, katerega barvni obseg so vse barve, ki jih je sposobno zajeti človeško oko. Poznamo več standardnih barvnih prostorov, vsak je značilen za določeno področje. Pri monitorjih se na primer zaradi njihovih lastosti uporablja aditiven RGB-prostor, medtem ko se pri tiskalnikih zaradi njihovih značilnosti uporablja subtraktiven CMYK-prostor. Med vsemi barvnimi prostori se lahko premikamo s pomočjo 3 x 3 matrike, ki definira preslikavo. Za primer vzemimo RGB-prostor. Za preslikavo iz XYZ-prostora uporabimo sledečo matriko:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 3.2405 & -1.5371 & -0.4985 \\ -0.9693 & 1.8760 & 0.0416 \\ 0.0556 & -0.2040 & 1.0572 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}. \tag{2.2}$$

X, Y, Z so vrednosti prostora XYZ. R, G in B so rezultati v prostoru RGB. Če bi na primer želeli izračunati $V(\lambda)$, ki je enak $Y(\lambda)$, uporabimo drugo vrstico inverzne matrike

$$V(\lambda) = Y = 0.2126R + 0.7152G + 0.0722B. \tag{2.3}$$



Slika 2.3: CIE 1931 2-stopinjski XYZ-standard, kjer so tri osnovne barve nastavljene tako, da ne prihaja do negativnih vrednosti in je $Y(\lambda)$ enaka $V(\lambda)$[12].

2.2 Kodiranje VDR-slik

Pomemben razmislek za vsako digitalno fotografijo je način shranjevanja. To še posebno velja za slike VDR, kjer je vsebovan veliko večji barvni razpon od 24-bitnega pri slikah NDR. Na srečo obstaja nekaj formatov za shranjevanje VDR-slik, ki so že izpopolnjeni.

Slike nizko dinamičnega razpona uporabljajo kodiranje, ki je prilagojeno prikazovanju na določeni napravi. Pravimo, da so izhodno orientirane. VDR-slike pa so nasprotno vhodno orientirane, torej obstaja neposredna povezava med senzornim elementom in svetlobnim sevanjem prizora. Barvno kodiranje seveda ne more shraniti barv izven svojega obsega, zato potrebujemo nove vrste shranjevanja. Za prikazovanje na NDR-prikazovalnikih pa nato sliko obdelamo s postopkom preslikave barvnih tonov. Le-ta zmanjša obseg dinamičnega razpona, pri tem pa strmimo k čim manjši izgubi informacije na sliki. Tako kljub manjšemu razponu slika človeškemu očesu še vedno prikazuje več detajlov, vsebuje več informacij in bolj verodostojno prikazuje prizor.

2.2.1 VDR-formati

Poznamo tri glavne VDR-formate, ki uporabljajo različne vrste kodiranja slik. Skupaj z glavnimi lastnostmi so prikazani v tabeli 2.3. Obstaja veliko načinov kodiranja VDR-slik. V tabeli 2.4 so našteti najbolj znani. Nekateri so uporabljeni v enem izmed formatov, drugi obstojajo le kot objavljeni standardi. Za primerjavo je podano tudi sRGB-kodiranje, ki se uporablja pri NDR-slikah. V prvem stolpcu je podano ime kodiranja. V drugem stolpcu se nahaja barvni prostor, ki ga kodiranje uporablja. Tretji stolpec vsebuje število bitov, ki jih kodiranje porabi za prikaz enega slikovnega elementa. Stolpec štiri vsebuje dinamični razpon, izračunan v logaritemskem merilu. Zadnji stolpec pa prikazuje največji relativni kvantizacijski korak med dvema sosednjima vrednostma, ki jih kodiranje lahko uporabi.

Tabela 2.3: Formati VDR-slik [10].

Ime	**Kodiranje**
HDR	RGB, XYZE
TIFF	IEEE RGB, LogLuv24, LogLuv32
EXR	Half RGB

Tabela 2.4: Kodiranja in njihove značilnosti [10].

Kodiranje	**Barvni prostor**	**bit/s.e.**	**Razpon (log_{10})**	**Rel. korak**
sRGB	RGB interval [0,1]	24	1.6 v.r.	variabilen
RGBE	Pozitivni RGB	32	76 v.r.	1.0 %
XYZE	CIE XYZ	32	76 v.r.	1.0 %
IEEE RGB	RGB	96	79 v.r.	0.000003 %
LogLuv24	$logY + (u', v')$	24	4.8 v.r.	1.1 %
LogLuv32	$logY + (u', v')$	32	38 v.r.	0.3 %
Half RGB	RGB	48	10.7 v.r.	0.1 %
scRGB48	RGB	48	3.5 v.r.	variabilen
scRGB-nl	RGB	36	3.2 v.r.	variabilen
scYCC-nl	YC_BC_R	36	3.2 v.r.	variabilen

Najbolj pomembni za ocenjevanje kakovosti kodiranja so podatki o porabi bitov na slikovni element (varčnost) in podatek o velikosti kvantizacijskega koraka. Prevelik korak se namreč opazi - barve na sliki se ne prelivajo, ampak se spreminjajo stopničasto. Da bi bila kvantizacija popolnoma nevidna, je potreben korak manjši od 1 %. To je cilj večine VDR-kodiranj, nekatere pa ga imajo še precej manjšega [10].

2.3 Prikazovanje VDR-slik

Naprave za prikazovanje delimo na dve veliki skupini: na tiskalnike in na prikazovalnike. Prva skupina vsebuje različne vrste tiskalnikov, od navadnih brizgalnikov, laserskih tiskalnikov do tradicionalnega tiska. Sam spadajo vse metode, ki preslikajo sliko na 2-dimenzionalni medij. Kategorija prikazovalnikov pa vsebuje navadne, katodne monitorje, LCD- in druge monitorje, projektorje in druge naprave, ki so sposobne prikazati sliko na 2-dimenzionalni površini. Prikazovalniki ponavadi uporabljajo lasten vir svetlobe, medtem ko se tiskalniki zanašajo na svetlobo iz okolja. Zaradi te poglavitne razlike so tudi problemi, ki se pojavljajo, precej različni.

2.3.1 Tiskalniki

Prve naprave za tiskanje so poznane že nekaj stoletij. Kljub razvoju tehnologije na tem področju imajo naprave za tiskanje nizek dinamični razpon. Za to sta predvsem dva razloga.

Prvi razlog je, da se za osvetlitev slike uporablja svetloba okolice. Tudi najsvetlejši bel papir, ki odbije večino svetlobe, ki pade nanj, ne more preseči količine svetlobe, ki je na voljo.

Druga omejitev je količina svetlobe, ki jo površina lahko absorbira. Tudi najboljše črnilo ne zmore absorbirati več kot 99.5 % svetlobe, zato prihaja do poslabšanja kontrasta v temnejših delih slike [10].

2.3.2 Prikazovalniki

Glavni člani skupine prikazovalnikov so monitor s katodno cevjo, LCD- in plazma prikazovalnik. Monitor s katodno cevjo ima omejen dinamični razpon zaradi največje energije, ki jo je še varno uporabiti, ne da bi poškodovali fosforni nanos in ne da bi ogrožali zdravje zaradi sevanja. Na drugi strani ima LCD-prikazovalnik drugačne težave. Svetlost navzgor ni omejena, kajti vedno je mogoče uporabiti močnejši vir svetlobe. Problem je pri prikazovanju temnih

delov slike, kajti kristalčki v prikazovalniku puščajo določen del svetlobe in tako omejujejo dinamični razpon [10]. Tudi plazma prikazovalniki imajo podoben problem, kajti vsak prikazovalni element mora biti pred osvetlitvijo napolnjen z električnim tokom, tako da tudi plazma prikazovalniki ne zmorejo prikazati prave črne. Imajo pa boljše rezultate kot LCD-zasloni, zato je tudi njihov razpon nekoliko večji.

Že nekaj let obstajajo prototipi prikazovalnikov z visokim dinamičnim razponom, ki pa so zaenkrat še predragi za splošno uporabo. Prvi VDR-monitor je leta 2005 pričelo prodajati podjetje BrightSide. Temelji na mreži led diod, ki nadomeščajo enoten vir svetlobe. Posamezna dioda tako oddaja določeno moč svetlobe ali pa je celo ugasnjena. Tako je odstranjen problem s puščanjem pri navadnih LCD-prikazovalnikih, saj ugasnjena dioda ne oddaja nikakršne svetlobe. Hkrati pa so posamezni deli slike različno osvetljeni, kar pripomore k še večjemu dinamičnemu razponu [13].

2.3.3 Preslikava barvnih tonov

Izraz preslikava barvnih tonov je poznan že iz analogne fotografije. Tam je predstavljal postopek prilagajanja osvetlitve slike pri razvijanju negativa.

V digitalni fotografiji ima izraz nekoliko drugačen pomen. Predstavlja ime za skupino algoritmov, ki sliko visokega dinamičnega razpona preslikajo v sliko nizkega dinamičnega razpona, pri tem pa strmijo k čim manjši izgubi informacij.

Ti algoritmi upoštevajo lastnosti človeških oči, zaznavanja možganov, psiholoških učinkov in vrsto drugi lastosti ter tako zgenerirajo čim bolj verodostojno sliko. Večina algoritmov uporablja določene modele vidnega sistema in tako v čim večji meri izkoriščajo danosti našega zaznavanja [10].

Področje preslikave barvnih tonov je preveč obširno in ni tema tega dela, zato se vanj ne bomo podrobneje spuščali. Ko bo potrebno bomo le omenili, kateri izmed algoritmov je bil uporabljen.

Poglavje 3

Zajemanje slik visokega dinamičnega razpona

Fotoaparat in senzor znotraj njega sta v svojem bistvu nepopolna naprava za merjenje sevanja svetlobe razporejene po prizoru. Nepopolna zato, ker ne more zajeti vsega spektralnega in dinamičnega razpona. Senzor je izpostavljen barvi in dinamičnemu razponu svetlobe, ki vstopa v fotoaparat skozi lečo, ki je pasiven element in tako le preusmeri svetlobo. Vse informacije so tako na voljo, vendar omejitve pri izdelavi senzorjev preprečijo, da bi se vse tudi uporabile.

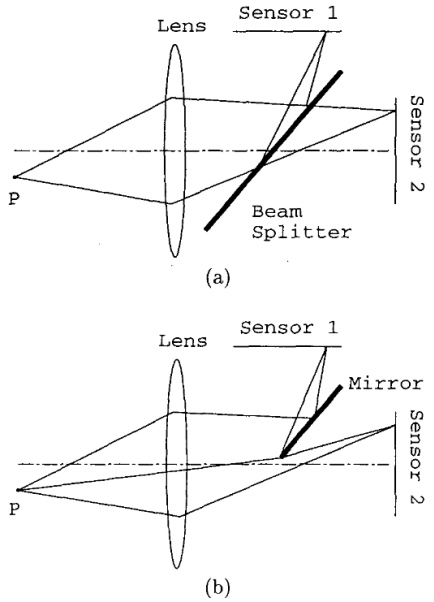
Analogne kamere s filmi pravzaprav zajamejo večji dinamični razpon kot novejše digitalne kamere. Standardna emulzija črno-belega filma lahko namreč doseže razpon skoraj štirih velikostnih redov. Razpon nekoliko zmanjšajo kakovost leče, odboji svetlobe na leči in kvaliteta črnine v notranjosti aparata [10].

Strokovnjaki so pristopili k reševanju problema zajemanja visokega dinamičnega razpona na več načinov. Delijo se na dva dela, na rešitve, ki z novimi tehnologijami rešujejo samo zajemanje informacij, in na rešitve, ki zajamejo več slik pri različnih časih izpostavitve, z navadnimi senzornimi elementi in aparati, nato pa le-te združijo v izboljšano sliko.

3.1 Sočasno zajemanje slik

Sočasno zajemanje slik poteka tako, da se svetloba razdeli, ko se odbije od leče fotoaparata, in tako zadene več različno občutljivih senzorjev. Svetlobo se lahko razdeli na več načinov, na primer s pomočjo posebnih prizem ali pa s polprosojnimi ogledali.

Obstajata dve slabosti tega pristopa. Prva je, da dodatna zrcala in leče zaradi svoje neidealnosti vnesejo dodatne nepravilnosti in popačenja v sliko, druga pa je ta, da večina razdelilnikov svetlobe zmore svetlobo razdeliti samo na dva snopa, tako da če želimo več slik, je potrebno uporabiti še več dodatnih elementov. To pa omejujejo prostorski problemi, predvsem zahteva po čim manjši poti do senzorja.



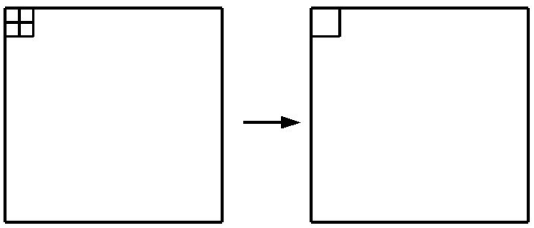
Slika 3.1: Dva načina razdelitve svetlobe: (a) razdelitev s polpresojnim ogledalom, (b) razdelitev le dela svetlobe z navadnim ogledalom [1].

Obema slabostma se lahko izognemo tako, da samo odprtino razdelimo na toliko delov, kot je senzorjev, svetlobo pa nato samo usmerimo na senzor s pomočjo zrcal. Slika 3.1 prikazuje skico obeh rešitev [1].

3.2 Več senzorskih elementov za zajem enega slikovnega elementa

Nov pristop k zajemanju slik visokega dinamičnega razpona predstavlja posebna vrsta senzorjev, kjer vsak slikovni element pravzaprav vsebuje več med seboj ločenih in različno občutljivih senzornih elementov. Vsak senzorni element izmeri moč sevanja svetlobe nanj, nato pa se vse meritve skupaj združijo

v rezultatu, slikovnem elementu [11]. Shema delovanja je prikazana na sliki 3.2.



Slika 3.2: Več senzorskih elementov se preslika v en slikovni element, pri tem se zmanjša ločljivost slike.

Ta tehnika sicer omogoča slike visokega dinamičnega dosega, vendar na račun ločljivosti slike, kajti slikovni element se sestavi iz več senzornih elementov, ki zasedajo tudi večji prostor. Sam postopek izdelave je trenutno drag in zapleten. Slabost je tudi združevanje rezultatov že v senzorju, saj mora biti kar se le da preprosto [9].

3.3 Drugačno merjenje svetlobnega sevanja

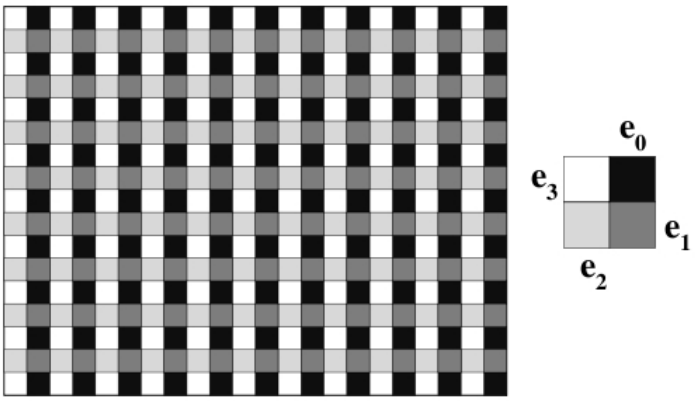
Obstajajo tudi senzorji z drugačnimi načini merjenja svetlobnega sevanja. Primer je senzor, ki namesto količine svetlobe, ki pade nanj, izmeri čas, ki je potreben, da se senzorni element zasiči. Ker je čas zasičenja za vse senzorne elemente enak, lahko rečemo, da je čas proporcionalen moči sevanja, ki pade na senzor. Izmerjene vrednosti so neto pretvorjene v sliko visokega dinamičnega razpona.

Problemi nastanejo s povečevanjem resolucije senzorja, kajti takšen senzorski element je velik in drag. Zaradi načina merjenja, ki pri merjenju majhne količine svetlobe traja dalj časa, je takšen senzor tudi manj odporen na zameglitev zaradi gibanja [2].

3.4 Prostorsko spreminjajoča izpostavitev senzornih elementov

Slika 3.3 prikazuje polje senzornih elementov, ki ponazarja prostorsko spreminjajočo izpostavitev. Štirje sosednji senzorni elementi imajo tako različne

izpostavitve, da velja $e_0 < e_1 < e_2 < e_3$. Bistvo tega pristopa je, da se nepretrgoma preverja tako prostorska kot tudi izpostavitvena dimenzija sevanja svetlobe. Primer, če je določen senzorni element zasičen, je velika verjetnost, da vsaj eden izmed sosednjih senzornih elementov ni zasičen. Enako je, če določen senzorni element prejme rezultat nič, ima najverjetneje eden izmed sosednjih senzornih elementov rezultat drugačen od nič.



Slika 3.3: Polje senzornih elementov različne občutljivosti [9].

Omeniti je potrebno, da pri izdelavi nismo omejeni s prikazano razporeditvijo senzornih elementov, ampak lahko uporabimo drugo, neponavljajočo ali celo naključno. Izdelava takšnega senzorja lahko poteka na več načinov. Eden izmed njih je, da prekrijemo polje senzornih elementov z mrežo različno prosojnih elementov.

Ta pristop je nekoliko boljši kot pristop z več senzorskimi elementi za en slikovni element. Resolucije slike se namreč ne zmanjša toliko, saj ostane število slikovnih elementov enako. Do manjše izgube prihaja le pri na primer zelo svetlih delih prizora, kjer so določeni senzorni elementi zasičeni in tako neuporabni [9].

Poglavje 4

Zajemanje VDR-slik z večkratno izpostavitvijo

Zajemanje VDR-slik z večkratno izpostavitvijo je s tehničnega vidika najpreprostejše, kajti ne potrebujemo nobene dodatne ali prilagojene opreme. Slike zaporedoma zajamemo z navadnim senzorjem, nato pa jih programsko združimo in obdelamo.

Zaradi zaporednega zajemanja slik nastopijo dodatne težave. Fotoaparat se lahko med zajemanjem slik premakne. Ravno tako se lahko objekti v prizoru slikanja premaknejo ali spremenijo. Zato je potrebno pri izdelavi končne slike porabiti dodane in bolj zapletene algoritme .

Splošni postopek generiranja slik visokega dinamičnega razpona iz več slik nizkega dinamičnega razpona poteka po naslednjih korakih:

1. Če za slikanje nismo uporabili stojala in daljinskega sprožilca, je najverjetneje prišlo do majhnih premikov pri zajemanju posamezne slike. Te premike je potrebno izničiti, torej moramo slike najprej poravnati.

2. Nato je potrebno poiskati funkcijo odziva senzorja. Le-ta nam omogoči združevanje slik v pravem medsebojnem razmerju.

3. Če imamo na sliki gibajoče se objekte, je potrebno, ko je funkcija odziva poznana, pred združitvijo slik uporabiti enega izmed algoritmov za odstranjevanje napak zaradi gibanja.

4. Slike s pomočjo izbrane utežne funkcije združimo.

5. Po združitvi odstranimo napake zaradi odbojev na leči.

4.1 Poravnava slik

Zaradi sekvenčnega zajemanja slik lahko nastane neporavnanost med posameznimi slikami. Dve tehniki, ki poskušata rešiti ta problem, sta predstavljeni v nadaljevanju.

4.1.1 Lucas in Kanade tehnika ocenjevanja gibanja

Prva tehnika rešuje tudi problem premikanja objektov znotraj slike. Osnovana je na variaciji Lucas in Kanade tehniki ocenjevanja gibanja [10]. Vsakemu slikovnemu elementu se izračuna vektor gibanja med posameznimi slikami. Ko je gibanje vseh slikovnih elementov izračunano, se slike izkrivijo, tako da jih je mogoče združiti v eno sliko. Prednost te tehnike je predvsem v tem, da zmore kompenzirati tudi zelo veliko gibanje. Ima pa tudi slabosti. Zahteva, da je funkcija odziva senzorja poznana in je precej računsko zahtevna.

4.1.2 Tehnika bitne slike mediane

Druga tehnika je preprostejša in ne zahteva poznavanje funkcije odziva [10]. Vhod v to tehniko je N 8-bitnih sivinskih slik. Kot njihov približek lahko uporabimo kar zeleni barvni kanal ali pa jih izračunamo s pomočjo enačbe

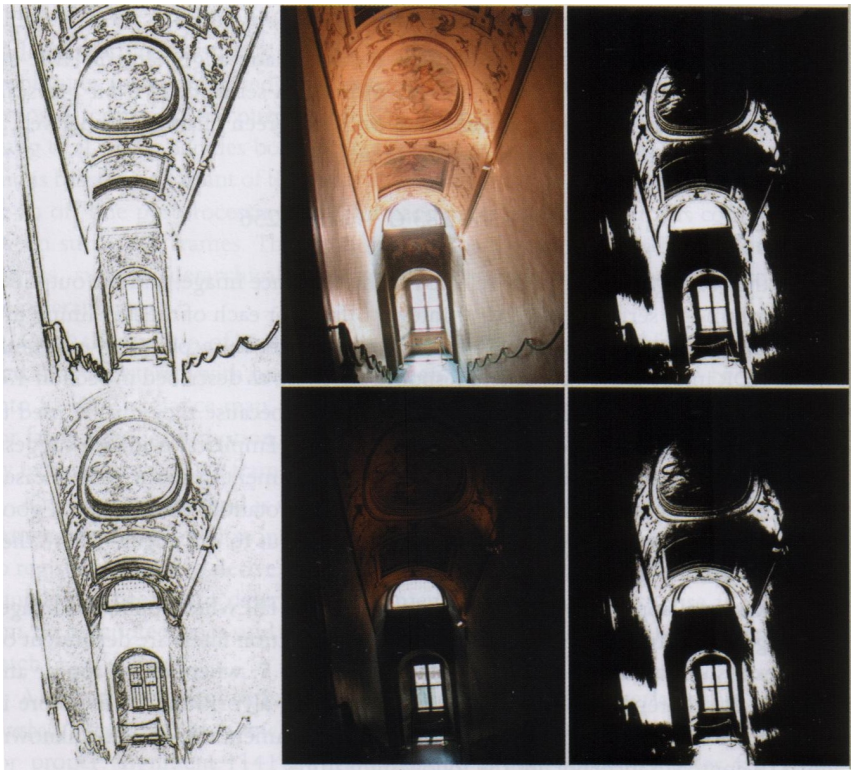
$$Y = (54R + 183G + 19B)/256. \tag{4.1}$$

Ena izmed slik je izbrana kot referenčna slika. Izhod iz algoritma je tako N-1 (x, y) celoštevilskih odmikov za vsako od preostalih slik. Postopek je sledeč:

1. Izračunamo 8-bitno mediano iz histograma slike.
2. Izdelamo bitno sliko, kjer je vrednost slikovnega elementa 0, če je vrednost vhodnega slikovnega elementa manjša ali enaka mediani in 1, ko je vrednost večja.
3. S pomočjo dobljenih bitnih slik poravnamo vhodne slike.

Za poravnavo slik se ponavadi uporabljajo algoritmi, ki uporabljajo iskanje robov. To pa pri slikah z različnim časom izpostavitve ni primerno, saj tukaj robovi niso enaki. Primer vidimo na sliki 4.1. Desno slika robov, v sredini originalna slika in levo bitna slika, dobljena s pomočjo mediane. To je srednja vrednost vseh slikovnih elementov na sliki. Z belo barvo so predstavljeni vsi

slikovni elementi z višjo vrednostjo, s črno pa vsi elementi z nižjo vrednostjo. Vidimo, da sta bitni sliki enaki, medtem ko se sliki robov razlikujeta.



Slika 4.1: Prikaz uporabe mediane in iskanja robov kot temeljev za poravnavo slike. Levo iskanje robov, sredina originalni sliki, desno mediana [10].

V določenih pogojih se lahko pojavijo pari slik, ki so pretemni ali pa presvetli, da bi lahko uporabili mediano. Takrat lahko za mejo uporabimo na primer 17. ali pa 83. percentil, torej vrednost, kjer je 17 oziroma 83 odstotkov slikovnih elementov svetlejših.

Ko so bitne slike izračunane, obstaja več načinov za poravnavo. Za uporabo najbolj preprost je algoritem, ki testira vse odmike znotraj dovoljenega obsega. Se pravi, izračuna logično operacijo XOR med obema slikama in določi odmik z minimalnimi odstopanji. Bolj učinkovita metoda je izračunavanje lokalnega minimuma, kjer izračunano le razlike od začetnega odmika (0,0) do najbližjega minimuma. Tretja metoda je uporaba slikovnih piramid. Ta metoda se začne s pomanjševanjem vhodnih slik do faktorja $log_2(najvecji\_odmik)$. Nato iz vsake velikosti izračunamo bitno sliko in potem referenčni odmik znotraj obsega ± 1. Iz vsake velikosti slik tako dobimo en bit odmika. Ta metoda

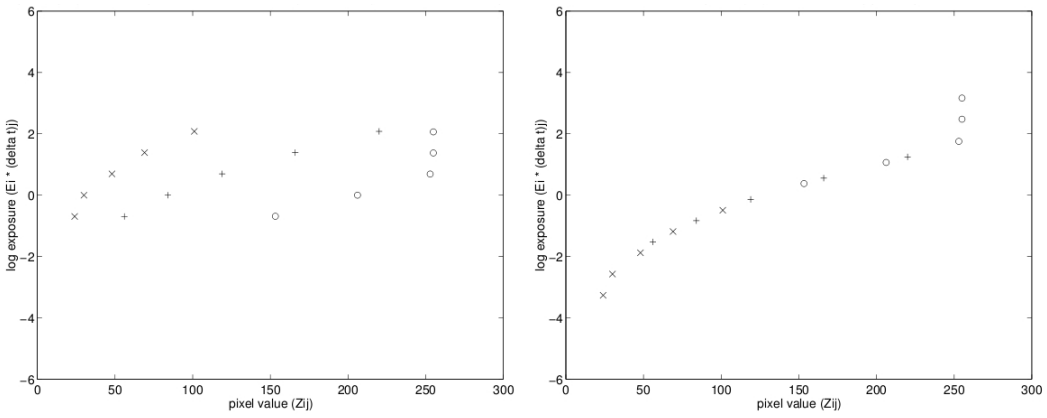
je podobno hitra kot iskanje lokalnega minimuma in ponavadi najde globalni minimum znotraj dovoljenega obsega [10].

4.2 Iskanje senzorjeve funkcije odziva

Za združitev NDR-slik v VDR-slike je potrebno poznati funkcijo odziva senzorja, tako da lahko lineariziramo podatke. V splošnem te funkcije proizvajalci ne želijo podati, tako da jo je potrebno izluščiti iz podatkov, ki so na voljo. Prikazali bomo dve tehniki, ki omogočata njen izračun.

4.2.1 Tehnika Debevca in Malika

Debevec in Malik sta predstavila preprost in robusten algoritem za iskanje funkcije odziva senzorja [3]. Osnovna ideja je, da z zajemanjem istega prizora pri različnih izpostavitvah pravzaprav selektivno vzorčimo funkcijo odziva. Na sliki 4.2. je prikazano to dogajanje za tri točke, zajete pri petih različnih izpostavitvah. Tako poznamo potek funkcije odziva v teh treh delih. Ker ne vemo, kako posamezni deli spadajo skupaj, sta Malik in Debevec uporabila linearno optimizacijo in tako našla gladko krivuljo, ki minimizira kvadratno napako preko funkcije odziva.



Slika 4.2: Primer iskanja funkcije odziva. Levo nelinearizirane točke, desno linearizirane točke [3].

Če vzamemo vrednost slikovnega elementa Z_{ij} z lokacije i in slike j, lahko zapišemo funkcijo obsevanosti E_i in časa izpostavitve Δt_j kot

$$Z_{ij} = f(E_i \Delta t_j). \tag{4.2}$$

Ker predvidevamo, da je funkcija monotona, je inverzibilna, torej lahko zapišemo

$$f^{-1}(Z_{ij}) = E_i \Delta t_j. \tag{4.3}$$

Če obe strani logaritmiramo in predpostavimo $g = lnf^{-1}$, lahko zapišemo

$$g(Z_{ij}) = lnE_i + ln\Delta t_j. \tag{4.4}$$

Iskanje funkcije torej pretvorimo v sistem enačb, kjer sta neznani vrednosti obsevanost E_i in funkcija g, za katero vemo, da je gladka in monotona. Če predpostavimo, da imamo P slik in na njih N slikovnih elementov ter da ima funkcija odziva $g(z)$ končno število vrednosti med Z_{min} in Z_{max} lahko problem pretvorimo na iskanje $(Z_{max} - Z_{min} + 1)$ vrednosti funkcij $g(Z)$ in N vrednosti lnE_i, ki minimizirajo sledečo objektivno kvadratno enačbo

$$O = \sum_{i=1}^{N} \sum_{j=1}^{P} [g(Z_{ij}) - lnE_i - ln\Delta t_j]^2 - \lambda \sum_{z=Z_{min}+1}^{Z_{max}-1} [w(z)g''(z)]^2. \tag{4.5}$$

Na novo se v tej enačbi pojavi utež $w(z)$, ki glede na vrednost slikovnega elementa določa stopnjo njegovega upoštevanja, in skalar λ, utež, ki določa, v kolikšni meri se člen upošteva glede na stopnjo šuma v podatkih. Prvi del enačbe služi zadostitvi pogojev v smislu najmanjše kvadratne napake, medtem ko drugi del enačbe s pomočjo drugega odvoda funkcije g poskrbi za gladkost prehodov. Za odvod uporabimo funkciji

$$g''(z) = g(z-1) - 2g(z) + g(z+1). \tag{4.6}$$

Gladilni člen je nujno potreben za lepo združevanje posameznih delov.

Dodati je potrebno še dve dopolnili, tako da je opis algoritma popoln in uporaben.

Prvič, $g(z)$ in E_i imata lahko vrednosti le do nekega skalarja α. Če g nadomestimo z $g+\alpha$ in lnE_i z $lnE_i+\alpha$ ne spremenimo objektivne enačbe. Da bi omejili velikostni faktor, dodamo pogoj $Z_{mid} = 0$, kjer je $Z_{mid} = \frac{1}{2}(Z_{min} + Z_{max})$. Pomen te omejitve je v tem, da bo imel slikovni element na sredini enotino izpostavitev.

Drugič, ker vemo, kakšno obliko krivulje pričakujemo, lahko uporabimo dodatne uteži za boljše prileganje funkcije. Uporabimo preprosto utežno funkcijo:

$$w(z) = \begin{cases} z - Z_{min} & za\ z \leq \frac{1}{2}(Z_{min} + Z_{max}) \\ Z_{max} - z & za\ z > \frac{1}{2}(Z_{min} + Z_{max}) \end{cases} \tag{4.7}$$

In tako dobimo končno enačbo

$$O = \sum_{i=1}^{N} \sum_{j=1}^{P} \{w(Z_{ij})[g(Z_{ij}) - lnE_i - ln\Delta t_j]\}^2 - \lambda \sum_{z=Z_{min}+1}^{Z_{max}-1} [w(z)g''(z)]^2. \tag{4.8}$$

Ni potrebno, da za iskanje funkcije odziva uporabimo vse slikovne elemente, ki so na voljo. Dovolj je, da zadostimo enačbi $N(P-1) > Z_{max} - Z_{min}$. Dobljeni problem rešimo z razcepom s singularnimi vrednostmi.

4.2.2 Tehnika Mitsunga in Nayarja

Misunga in Nayar sta ravno tako predstavila pristop k iskanju funkcije odziva senzorja. Funkcijo odziva sta pridobila s pomočjo polinomske aproksimacije. Definirala sta sledečo N-dimenzionalno polinomsko enačbo [8]

$$f(Z) \ = \sum_{n=0}^{N} c_n Z^n, \tag{4.9}$$

kjer je $f(Z)$ funkcija odziva senzorja na sevanje svetlobe Z. Tako je končna funkcija odziva definirana z $N+1$ koeficienti tega polinoma $\{c_0, ...c_N\}$. Da bi našli te koeficiente, je potrebno sledečo funkcijo napake za dane kandidate razmerja izpostavitve, $R_{j,j+1}$, minimizirati

$$\varepsilon = \sum_{i=1}^{I-1} \sum_{j=1}^{J-1} \left[\sum_{n=0}^{N} C_n Z_{i,j}^n - R_{j,j+1} \sum_{n=0}^{N} c_n Z_{i,j+1}^n \right]. \tag{4.10}$$

Funkcija sešteva preko vseh točk i in preko vseh slik j. C_n so prej omenjeni koeficienti polinoma, $Z_{i,j}$ je vrednost slikovnega elementa na sliki j, $Z_{i,j+1}$ pa vrednost enakoležečega slikovnega elementa na sliki $j+1$. $R_{j,j+1}$ pa je razmerje med dvema izpostavitvima j in $j+1$. Minimum najdemo tako, da rešimo sistem $N+1$ enačb oblike

$$\frac{\partial \varepsilon}{\partial c_n} = 0. \tag{4.11}$$

Kot v prejšnji metodi je tudi tukaj potrebno dodati pogoj $f(1) = 1$, le-ta pa zmanjša število dimenzij za ena, saj velja:

$$c_N = 1 - \sum_{n=0}^{N-1} c_n \tag{4.12}$$

Da bi izračunali prava razmerja med izpostavitvami, je uporabljena iterativna metoda, kjer je prikazan sistem enačb ponavljajoče rešen, med vsako ponovitvijo pa se razmerja popravijo po formuli:

$$R_{j,j+1}^{(k)} = \sum_{i=1}^{I} \frac{\sum_{n=0}^{N} C_n^{(k)} Z_{i,j}^n}{\sum_{n=0}^{N} C_n^{(k)} Z_{i,j+1}^n} \tag{4.13}$$

Ponovitve si sledijo, dokler se razmerja ne spreminjajo več veliko.

Mitsunga in Nayar sta upoštevala le sosednji izpostavitvi, bolje pa je, če se upoštevajo vse, saj je tako sistem bolj stabilen [10].

4.3 Združevanje slik

Ko je funkcija odziva senzorja znana in tako lahko lineariziramo podatke, je potrebno le še sestaviti slike v eno VDR-sliko. Pojavi se vprašanje, kakšne uteži $w(z)$ naj se uporabijo pri združevanju slik. Jasno je, da je potrebno slikovne elemente, ki so nasičeni ali pa popolnoma temni, torej enaki 0, izpustiti. Drugje je potrebno uporabiti eno izmed utežnih funkcij.

Najbolj osnovno sta predlagala Mann in Picard, kjer predlagata kar uporabo odvoda funkcije odziva senzorja $f(z)$, saj naj bi bili slikovni elementi, ki imajo višje vrednosti, tudi bolj zanesljivi [10].

$$w(z) = f'(z). \tag{4.14}$$

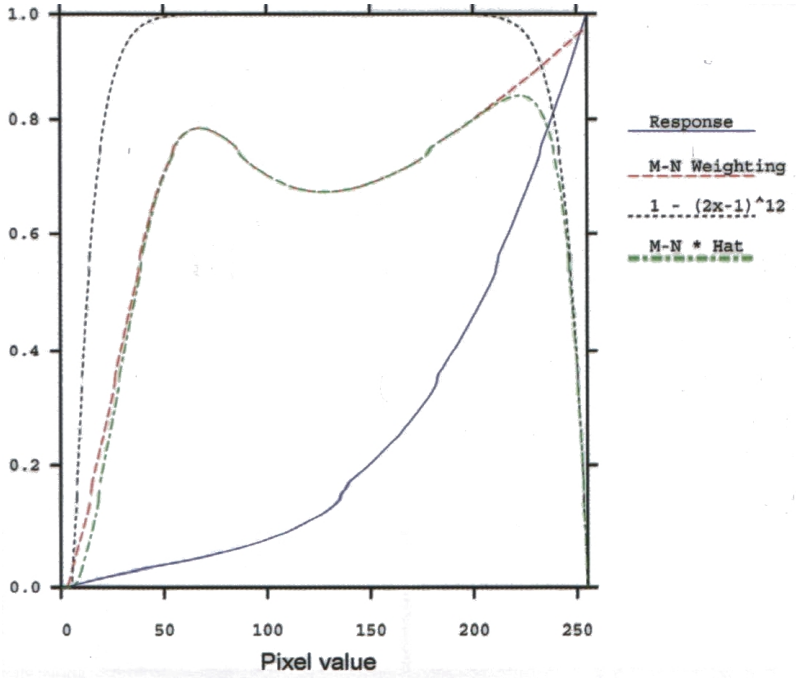
Debevec in Malik sta uporabila preprosto funkcijo v obliki klobuka, saj naj bi bili slikovni elementi s sredinskimi vrednostmi najbolj verodostojni. Primer je recimo [3].

$$w(z) = 1 - (2z - 1)^{12}). \tag{4.15}$$

Mitsunaga in Nayar sta s pomočjo teorije signalov predlagala deljenje funkcije odziva s svojim odvodom [8].

$$w(z) = \frac{f(z)}{f'(z)}. \tag{4.16}$$

Neglede na izbiro uteži bi moral biti rezultat zadovoljiv. Slika 4.3 prikazuje vse predstavljene poteke uteži [10].



Slika 4.3: Uteži, primerne za združevanje NDR-slik v VDR-slike [10].

4.4 Odstranjevanje napak zaradi gibanja objektov

Ko slike poravnamo in je znana funkcija odziva senzorja, lahko slike združimo. Vendar to velja samo za slike, kjer ni nikakršnih gibajočih objektov. Če se kakršenkoli objekt giblje po prizoru, ki ga slikamo, se na združeni sliki pojavi kot nekakšen polprosojen "duh". Ostaja več različnih načinov za odstranjevanje teh napak.

4.4.1 Ocenjevanje gibanja

Prva skupina tehnik se opira na algoritme za ocenjevanje gibanja. Za vsak slikovni element posebej se izračuna vektor gibanja, nato pa se slika izkrivi glede na ocenjene vektorje. Takšna izkrivljena slika, ki naj bi imela vse slikovne elemente na pravem mestu, je nato uporabljena za združevanje v VDR-sliko.

Te tehnike delujejo pravilno, če algoritmi za ocenjevanje gibanja opravijo svoje delo dobro. To pa jim velikokrat ne uspe, saj so ti algoritmi zanesljivi le do določene mere. Na primer, imajo slabe rezultate pri odsevnih in prosojnih

objektih. Problemi nastanejo tudi z izrisovanjem ozadja, kjer je bil poprej objekt. Ti problemi se nato prepoznajo tudi v končni sliki [10, 7].

4.4.2 Uporabe posameznih slik

Druga skupina tehnik uporablja pristop brez ocenjevanja gibanja. Predvideva se, da se posamezni gibajoč objekt nahaja znotraj dinamičnega razpona ene same slike in se tako za njegovo predstavitev uporabijo le informacije te slike.

Najprej je potrebno območja na sliki, kjer najverjetneje prihaja do gibanja, prepoznati. To naredimo z izračunom obtežene variance vrednosti slikovnih elementov za vse lokacije na sliki, nato pa izberemo območja, katerih varianca je nad določeno mejo. Za vsako področje posebej je nato izbrana slika, katere izpostavitev najbolje prikazuje to področje in podatke s te slike neposredno vključimo v VDR-sliko [10].

Ta tehnika sicer dobro deluje pri lahko ločljivih objektih, odpove pa pri objektih z zelo podobno barvo, kot jo ima ozadje.

Namesto variance se lahko uporabi mera, izpeljana iz entropije, saj je ta neodvisna od kontrasta med objektom in ozadjem. Nato ravno tako izberemo posamezno sliko, ki jo uporabimo za dani objekt [6].

Obe metodi delujeta relativno dobro, vendar odpovesta pri območjih z visokim dinamičnim obsegom, kajti uporaba le ene slike nizkega dinamičnega obsega onemogoči povečevanje obsega. Če vsebuje premikajoči objekt velik dinamični razpon, le-tega izgubimo [10, 6].

4.4.3 Neposredno spreminjanje uteži

Novejša tehnika, ki so je odkrili Khan in sodelovci, temelji na popolnoma novem pristopu k odstranjevanju napak zaradi gibanja.

V nasprotju s poprej omenjenimi načini ta tehnika ne uporablja neke vmesne stopnje, kot je na primer ocenjevanje gibanja, detekcija objektov in računanje variance ter entropije. Namesto tega se VDR-slika zgenerira neposredno iz podatkov NDR-slik, tako da se iterativno spreminjajo uporabljene uteži pri posamezni točki vsake slike glede na verjetnost, da je ta točka del statičnega dela slike (ozadja) in je pravilno izpostavljena.

Za izračun je uporabljen neparametrični model ozadja, zato niso potrebne nikakršne omejitve glede vrste ozadja. Edina predpostavka je, da je v sekvenci izpostavitev ozadje dominantno, torej se za vsako lokacijo predvideva, da je število slikovnih elementov, ki zajemajo ozadje, večje kot število slikovnih elementov, ki zajemajo premikajoči objekt.

Začetne uteži se izračunajo glede na vrednost posameznega slikovnega elementa kot ponavadi. Nato pa se uteži iterativno prilagodijo glede na rezultate izračuna modela ozadja. Končni rezultat je slika brez napak oziroma "duhov"[7].

4.5 Odstranjevanje odbojev svetlobe na leči

Po odstranitvi napak zaradi gibanja lahko na sliki ostanejo določeni artefakti zaradi neidealnih lastnosti optičnih delov fotoaparata. Večina fotoaparatov uporablja optične elemente, ki so predvideni za delo s senzorji nizkega dinamičnega razpona. Ti elementi pa niso dovolj kakovostni za VDR-slike, zato nastajajo nepravilnosti, na primer svetlobni odboji.

Teh nepravilnosti ne popravljamo na NDR-slikah, saj se preko posameznih NDR-slik le seštejejo, tako da jih lahko naknadno popravimo na že dobljeni VDR-sliki.

4.5.1 Funkcija širjenja svetlobne točke

Funkcija širjenja svetlobne točke je idealizirana radialno simetrična karakterizacija svetlobnih nepravilnosti, ki nastanejo okoli točkastega vira svetlobe v popolnoma temnem okolju (slika 4.4). Če to funkcijo poznamo, jo lahko uporabimo za popravljanje napak odbojev svetlobe. Ker se lahko lastnosti leče, na primer zaradi prahu, spreminjajo in je sam postopek zapleten, ga uporabljamo samo za najbolj kritične aplikacije.

Drugje pa lahko funkciji širjenja svetlobne točke najdemo približek iz same VDR-slike.

4.5.2 Iskanje približka funkcije širjenja svetlobne točke

Funkcija širjenja svetlobne točke definira, kako se svetloba razprši okoli svetlih točk na sliki. Da bi našli približek te funkcije, je potrebno izmeriti vse minimalne vrednosti slikovnih elementov okoli teh svetlih točk. Za dosego tega je potrebno sešteti vse potencialne vplive svetlih slikovnih elementov na določeni razdalji do temnejših slikovnih elementov in tako dobimo oceno funkcije, krožnico za krožnico.

Na primeru slike 4.5 vidimo sliko s tremi svetlimi slikovnimi elementi. Vsak slikovni element obkroža enaka krožnica, kar ustvari prekrivajoče se kroge. Če bi funkcijo širjenja poznali, bi preprosto pomnožili vse točke na teh krožnicah s funkcijo in tako dobili pravilne rezultate. Če delimo najtemnejši slikovni

Slika 4.4: Primer slike funkcije širjenja svetlobne točke [10].

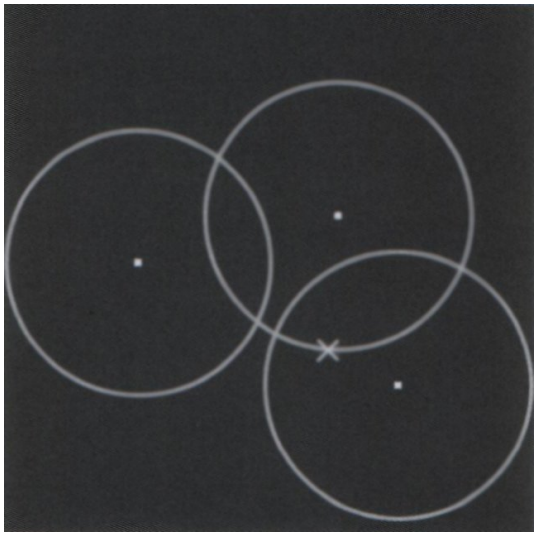
element na krogu s centrom kroga, dobimo zgornjo mejo funkcije. Če isti slikovni element prekriva več krožnic, je potrebno rezultate posameznih radijev sešteti.

Postopek iskanja funkcije širjenja je takšen:

1. Za vsako krožnico, ki jo upoštevamo:
   (a) Seštej vrednosti svetlih slikovnih elementov na določeni krožnici v posebno sivinsko sliko.
   (b) Najdi minimalno razmerje za vsoto temen_element/svetel_element preko vseh krožnic.
2. Če je minimum večji kot pri prejšnji, manjši krožnici, ga spreglej, saj predpostavljamo, da je funkcija monotona.
3. Za vsak slikovni element z minimalnim razmerjem, dobljen za vsako krožnico, upoštevaj vse doprinose preko celotne slike s spodnjo formulo

$$P_i = \sum_j P_j \left(C_0 + \frac{C_1}{r_{ij}} + \frac{C_1}{r_{ij}^2} + \frac{C_1}{r_{ij}^3} \right). \tag{4.17}$$

V tej formuli P_i predstavlja vrednost minimalnega slikovnega elementa, P_j vrednost drugih vplivajočih slikovnih elementov preko slike in r_{ij} predstavlja



Slika 4.5: Primer slike s tremi svetlimi slikovnimi elementi [10].

razdaljo med minimalnim slikovnim elementom ter vplivajočim slikovnim elementom. Štirje neznani parametri (C_0, C_1, C_2, C_3) so nato z lahkoto izračunani iz sistema enačb, če najdemo več kot 3 minimalne slikovne elemente [10].

4.5.3 Postopek odstranjevanja odbojev svetlobe na leči

Z dobljenim približkom funkcije širjenja svetlobne točke je sam postopek odstranitve odbojev precej preprost. Za vsak svetel slikovni element na sliki odštejemo njeno vrednost pomnoženo s funkcijo širjenja od okolice. Ker je to zelo zahtevna operacija, lahko uporabim zmanjšano ločljivost slike brez velikih izgub.

Koraki odstranjevanja so naslednji:

1. Ustvari zmanjšano sliko I_{CR} in dodatno sliko odbojev F_{CR} ter jo inicializiraj na črno.

2. Za vsak svetel slikovni element v I_{CR} sliko pomnoži s funkcijo širjenja svetlobne točke in rezultat prištej F_{CR}.

3. Če je vrednost kateregakoli slikovnega elementa v F_{CR} večja kot vrednost ustrezajočega slikovnega elementa v I_{CR}, primerno manjšaj amplitudo F_{CR}.

4. Povečaj F_{CR} z uporabo linearne interpolacije in jo odštej od originalne VDR-slike.

Korak tri poskrbi, da ni mogoče dobiti negativnih slikovnih elementov v izhodu algoritma. Zaradi interpolacije pa je še vedno mogoče, da v koraku štiri dobimo negativno vrednost slikovnega elementa, ki ga omejimo na 0. Slika 4.6 prikazuje rezultate algoritma skupaj s sliko odbojev, dobljeno v koraku 2 [10].

Slika 4.6: Slika okna pred odstranitvijo odbojev (levo) in po odstranitvi (sredina). Desna slika prikazuje vpliv funkcije širjenja svetlobne točke [10].

Poglavje 5

Implementacija

5.1 Razvojno okolje

Pri razvoju implementacije algoritmov za zajemanje slik visokega dinamičnega razpona z večkratno izpostavitvijo je bila uporabljena vrsta GNU/Linux operacijskega sistema, natančneje Ubuntu 8.04 LTS. Samo programiranje je potekalo v urejevalniku besedil VIM, nasledniku enega najbolj znanih urejevalnikov VI. Uporabljen programski jezik je C. Za prevajanje se je uprabil prevajalnik GCC 4.2.3. Pri razvoju implementacije sta bili uporabljeni dve že obstoječi programski knjižnici:

- libnetpmb-knjižnica, ki vsebuje funkcije in metode za delo s PPM-formatom slik [15].
- GSL - GNU Scientific Library, numerična knjižnica s preko 1000 funkcijami in metodami za vsa področja numeričnega računanja [16].

Po končni uporabi algoritmov, opisanih v tem poglavju, se je za nadaljnjo obdelavo in pripravo za prikaz na NDR-prikazovalnikih uporabila zbirka orodij exrtools [18] in aplikacija Qtpfsgui [17]. Orodja exrtools so preprosta v ukazni vrstici delujoča zbirka za delo z visoko dinamičnimi slikami formata EXR. Uporabljena so bila za prenašanje VDR-slike iz PPM-formata v EXR-format. Aplikacija Qtpfsgui je namenjena prav uporabi z VDR-slikami, omogoča generacijo, urejanje VDR-slik in ima veliko zbirko implementiranih algoritmov za preslikavo barvnih tonov. Prav z namenom preslikovanja barvnih tonov in s tem priprave na prikaz na NDR-prikazovalnikih smo jo tudi uporabili.
Za prevajanje in povezovanje programa (genHDRppm) je uporabljen ukaz

gcc genHDRppm.c genHDRppm.h /usr/lib/libnetpbm.a /usr/lib/libgsl.so /usr/lib/libgslcblas.so -o genHDRppm,

medtem ko je za zagon programa uporabljena sledeča struktura vhodnih parametrov

genHDRppm num_of_input_images input_image1.ppm exp_time1 input_image2.ppm exp_time2 [input_image3.ppm exp_time3...] lamba w_Sel flare_removal flare_output output_image.ppm.

Najprej navedemo število NDR-slik, ki jih bomo uporabili (num_of_input_images), nato navedemo njihova imena v paru z njihovimi časi izpostavitve v μs (input_image1.ppm exp_time1). Nato navedemo utež lambda, ki vpliva na združevanje NDR-slik v VDR-sliko. Nato je na vrsti izbira uteži, ki se uporabijo pri združevanju slik. Z vrednostjo 0 izberemo utežno funkcijo Debevca in Malika (4.7), medtem ko z 1 izberemo utežno funkcijo v obliki klobuka (4.15). Naslednja dva parametra vplivata na odstranjevanje napake zaradi osvetlobnih odbojev na leči. Prvi je flare_removal, ki z vrednostjo 0 izključi odstranjevanje napake, medtem ko vrednost 1 vključi algoritem za odstranjevanje napake odbojev. Drugi parameter, flare_output, pa, če ima vrednost 1 na izhod, namesto prave slike pošlje sliko samih svetlobnih odbojev na leči. Nazadnje navedemo ime izhodne datoteke za shranjevanje VDR-slike (output_image.ppm).

5.2 Osnovna zgradba implementacije

Pri zajemanju testnih slik je bilo z uporabo stojala in daljinskega sprožilca poskrbljeno, da so posamezne slike zajele popolnoma enak prizor in da med njimi ni prihajalo do neporavnanosti. Prav tako na slikah ni nikakršnih gibajočih se objektov. Zaradi naštetega pri sami implementaciji ni bilo smiselno implementirati funkcij poravnave slik in odstranjevanja napak zaradi gibanja, saj bi bil njihov doprinos h kakovosti rezultatov ničen in jih zato ne bi bilo mogoče vrednotiti.

Implementacijo algoritmov za zajemanje VDR-slik z večkratno izpostavitvijo tako lahko razdelimo na 4 dele:

- Funkcije za branje slik iz datotek in zapisovanje slik nazaj v datoteke.
- Funkcijo za iskanje funkcije odziva senzorja.
- Funkcijo za združevanje NDR-slik v eno-VDR sliko.

- Funkcijo za odpravljanje svetlobnih odbojev leče.

5.3 Branje in zapisovanje slik

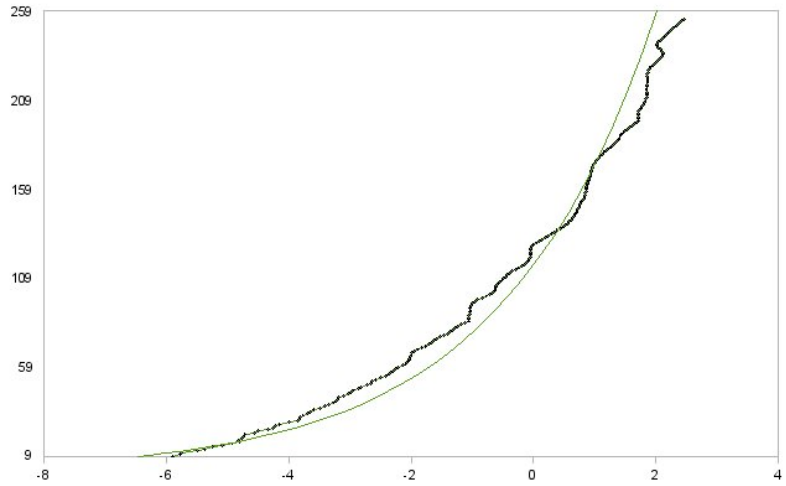
Implementacija pri svojem delovanju prejme kot parameter več vhodnih datotek s slikami. Uporabljen je format PPM, saj zaradi svoje preproste zgradbe omogoča lahko uporabo [15].

Za upravljanje s slikami v formatu PPM je uporabljena knjižnica libnetpbm, ki vsebuje funkcije in metode tako za branje slik iz datotek, za njihovo urejanje in nenazadnje tudi za shranjevanje slik v datoteke formata PPM.

Samo delovanje algoritmov je neodvisno od izbranega formata shranjevanja slik. Za uporabo drugačnega formata slik bi bilo potrebno le prilagoditi funkcije branja in zapisovanja.

5.4 Iskanje funkcije odziva senzorja

Za iskanje funkcije odziva senzorja je v implementaciji uporabljena tehnika Debevca in Malika [3], opisana v podpoglavju 4.2.1. Tehnika predpostavlja, da z zajemanjem istega prizora pri različnih izpostavitvah selektivno vzorčimo funkcijo odziva senzorja. Ker ne vemo, kako posamezni vzorčeni slikovni elementi spadajo skupaj, uporabimo linearno optimizacijo in tako najdemo iskano funkcijo (4.8). Primer grafov funkcije vidimo na sliki 5.1.



Slika 5.1: Primer grafov funkcije odziva senzorja.

Za rešitev problema iskanja funkcije potrebujemo vsaj toliko izbranih slikovnih elementov, da zadostimo pogoju $N(P-1) > Z_{max} - Z_{min}$, kjer N pomeni število slikovnih elementov, P število slik, Z_{min} in Z_{max} pa najmanjšo in največjo vrednost, ki jo lahko ima slikovni element. Pri uporabljenih slikah, ki imajo vrednosti od $Z_{min} = 0$ do $Z_{max} = 255$, torej potrebujemo vsaj 256 slikovnih elementov preko vseh slik. Ker pa posamezni slikovni elementi lahko zavzemajo enake vrednosti, je bolje, da jih uporabimo nekoliko več. Pri implementaciji se uporablja spremenljiva vrednost uporabljenih slikovnih elementov na sliko glede na število vhodnih slik, izračunana po formuli

$$N = floor(\frac{(Z_{max} - Z_{min}) \cdot 2}{P} + 1). \tag{5.1}$$

Skupna vrednost je tako, neglede na število vhodnih slik P, vedno dvakrat večja, kot zahteva pogoj.

Čas izpostavitve posameznih slik Δt_j je podan v μs preko vhodnih podatkov . Pred uporabo ga je potrebno logaritmirati in normirati z izpostavitvijo $1s = 1.000.000\mu s$

$$\Delta t_{log} = log_2(\Delta t) - log_2(1.000.000). \tag{5.2}$$

V implementaciji so uporabljene uteži $w(z)$, ki jih priporočata Debevec in Malik in so definirane v formuli (4.7).

Edina prava spremenljivka, poleg števila uporabljenih slikovnih elementov oziroma števila vhodnih slik, je v tem algoritmu utež λ, ki določa v kolikšni meri se upošteva gladilni člen. Prilagajamo jo glede na stopnjo šuma v vhodnih slikah.

Algoritem za iskanje funkcije odziva senzorja smo tako preizkusili pri različnih vrednostih spremenljivke λ in različnem številu vhodnih slik. Rezultati so predstavljeni v poglavju 6.

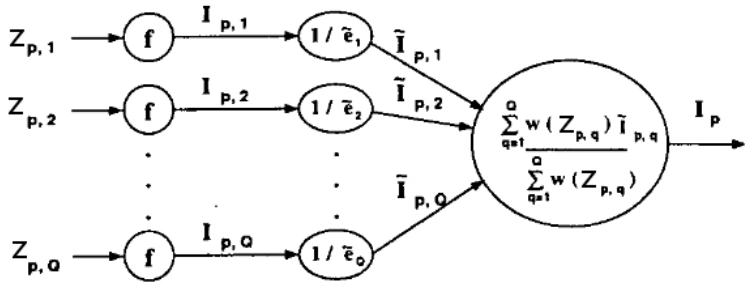
5.5 Združevanja NDR-slik v VDR-sliko

Združevanje slik nizkega dinamičnega razpona v eno sliko visokega dinamičnega razpona je jedro celotne implementacije. Sam postopek ni zapleten, grafično ga prikazuje slika 5.2. Za vsak slikovni element VDR-slike posebej si sledijo koraki:

1. Slikovni elementi NDR-slik so preslikani v vrednost sevanja svetlobe s pomočjo funkcije odziva senzorja.

2. Izračunane vrednosti sevanja svetlobe so normalizirane s časom njihove izpostavitve, tako da vsi slikovni elementi uporabljajo enako efektivno izpostavitev.

3. Slikovni elementi so združeni s pomočjo obteženega povprečenja.



Slika 5.2: Postopek združevanja slikovnih elementov posameznih slik v slikovni element VDR-slike [8].

Rezultati postopka so odvisni od izbrane vrste uteži. V tem diplomskem delu sta uporabljeni dve preprosti funkciji. Prvo sta podala Debevec in Malik [3] in je uporabljena že v podpoglavju 4.2.1 (4.7), druga je podana kot primer funkcije v obliki klobuka (4.15) v podpoglavju 4.3. Rezultati in primerjave so podane v poglavju 6.

5.6 Odstranjevanja svetlobnih odbojev leče

Poleg glavnega dela, iskanje funkcije odziva senzorja in združevanje slik, implementacija vsebuje tudi algoritem za odpravo napak odbojev svetlobe na leči, opisan v podpoglavju 4.5.

Ker je sam algoritem računsko zelo zahteven, se najprej ustvarita dve pomanjšani sliki, ena sivinska in ena barvna. Takšni sliki, ki imata širino 128 slikovnih elementov, se nato uporabita v nadaljnjem postopku.

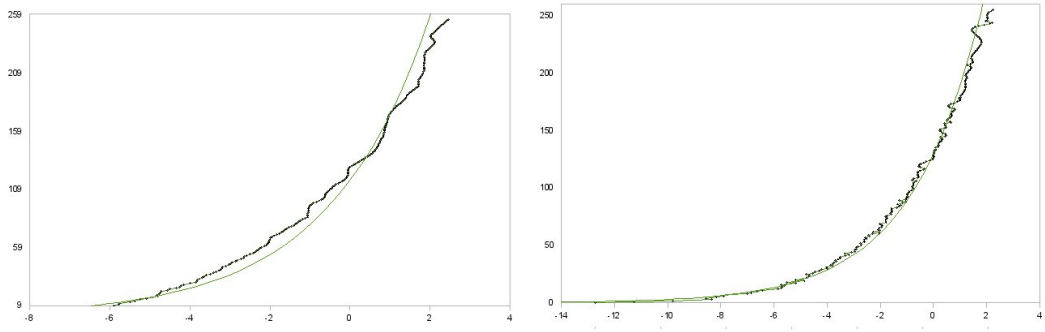
Za izračun približka funkcije širjenja svetlobne točke se uporabi razdalja krožnic med 3 in 64. Kot svetel slikovni element se vzame vse slikovne elemente, ki so vsaj 1000-krat večji od minimalnega slikovnega elementa.

Poglavje 6

Rezultati

6.1 Prilagajanje parametrov iskanja funkcije odziva senzorja

Algoritem iskanja funkcije odziva smo preizkusili pri različnih vrednostih uteži λ in različnem številu vhodnih slik. Tako smo dobili množico grafov, ki s svojo natančnostjo nakazujejo pravilno izbiro obeh parametrov algoritma. Na sliki 6.1 sta prikazana grafa najslabše in najboljše meritve zelenega barvnega kanala.



Slika 6.1: Graf najslabše in najboljše aproksimacije funkcije odziva senzorja.

Tabeli 6.1 in 6.2 kažeta vrednost napak za prvo serijo meritev pri različnem številu slik in različno izbrani vrednosti za utež lambda. Spomočjo kvadratne napake

$$\frac{1}{N}\sum_{n=0}^{N}(x - x_i)^2 \tag{6.1}$$

Tabela 6.1: Kvadratna napaka iskanja funkcije odziva senzorja pri različnem številu slik in različnih vrednostih uteži λ.

Št. slik | λ	**1.00**	**0.75**	**0.50**	**0.25**
2 - nizek razpon	0.38	0.37	0.35	0.35
2 - visok razpon	0.20	0.19	0.19	0.20
4	0.18	0.18	0.18	0.17
8	0.08	0.08	0.08	0.07

Tabela 6.2: Minmax napaka iskanja funkcije odziva senzorja pri različnem številu slik in različnih vrednostih uteži λ.

Št. slik | λ	**1.00**	**0.75**	**0.50**	**0.25**
2 - nizek razpon	5.10	4.96	4.48	4.09
2 - visok razpon	4.08	3.82	3.79	3.78
4	4.06	4.03	3.93	3.80
8	2.64	2.37	2.04	1.72

in minmax napake

$$minmax_{n=0:N}|x - x_i|, \tag{6.2}$$

kjer x predstavlja rezultat, x_i pa idealno vrednost, se izkaže, da je bistveno bolj pomembno samo število slik kot pa vrednost uteži lambda. Več kot imamo slik, boljše rezultate dobimo. Pokaže se, da je pomemben tudi razpon časov izpostavitve. Opravili smo namreč meritve z dvema slikama, ki imata podoben čas izpostavitve, in z dvema slikama z bolj oddaljenima časoma izpostavitve. Sliki z višjim razponom časov razporeditve sta imeli boljše razultate.

Iz rezultatov je tudi razvidno, da ima gladenje s pomočjo uteži λ, če je prekomerno, negativen učinek na natančnost izračunane funkcije odziva senzorja, saj so pri večjih vrednostih uteži λ tudi večje vrednosti napak. Da bi to bolje ovrednotili, je bila opravljena dodatna serija meritev, kjer je bilo število slik konstantno 8, utež λ pa je zajela širši razpon vrednosti. Vrednosti napak prikazuje tabela 6.3.

Tudi v drugi seriji meritev se izkaže, da previsoka vrednost uteži λ doseže slabše rezultate. Iz tega lahko sklepamo, da testne slike ne vsebujejo šuma, saj naj bi bila višja vrednost uteži λ potrebna predvsem pri slikah s šumom.

Tabela 6.3: Napake iskanja funkcije odziva senzorja pri 8 slikah in različnih vrednostih uteži λ.

Napaka | λ	**4.00**	**2.00**	**1.00**	**0.75**	**0.50**	**0.25**	**0.10**
Kvadratna napaka	0.11	0.10	0.08	0.08	0.07	0.07	0.07
Minmax napaka	3.45	3.19	2.64	2.37	2.04	1.72	1,61

Pri vseh nadaljnjih zagonih implementacije se je uporabljala vrednost $\lambda = 0.1$

6.2 Uporaba različnih vrst uteži pri združevanju slik v VDR-sliko

Za združevanje NDR-slik v VDR-sliko sta v implementaciji na voljo dve utežni funkciji. Utežna funkcija, ki sta jo predlagala Debevec in Malik, in funcija v obliki klobuka. Obe imata zadovoljive rezultate. Do razlik prihaja le pri zelo temnih delih slike, kjer je na sliki, generirani s pomočjo utežne funkcije klobuka, nekoliko manj šuma. Na sliki 6.2, ki vsebuje serijo NDR-slik in dve VDR-sliki generirani z obema utežnima funkcijama, so vidne razlike predvsem na garažnih vratih (levi del slike).

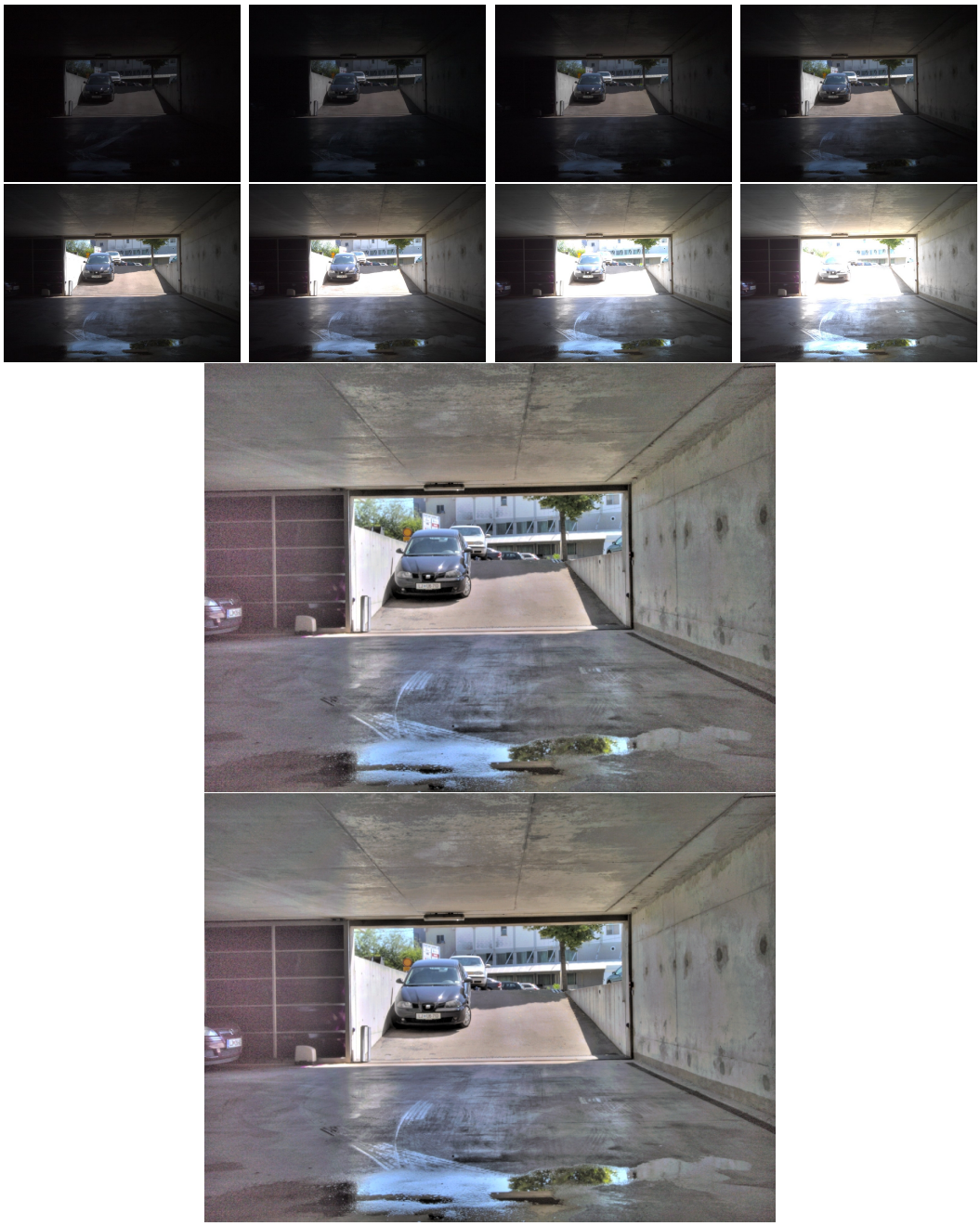
6.3 Izboljšanje dinamičnega razpona slike

Osrednji namen tega diplomskega dela je bil preizkusiti delovanje enega izmed algoritmov za izdelavo slik visokega dinamičnega razpona z večkratno izpostavitvijo. Vsem slikam se je dinamični razpon bistveno zvišal. Prejšnji razpon 1:256 je dosegel pri nekaterih serijah slik dinamični razpon preko 1:30000. V nadaljevanju slike 6.3, 6.4, 6.5 in 6.6 prikazujejo testne serije NDR-slik in generirano VDR-sliko.

6.4 Izboljšanje zaradi odstranjevanja svetlobnih odbojev leče

Algoritem za odstranjevanje svetlobnih odbojev deluje dobro. Pravilno izračuna približek funkciji širjenja svetlobne točke in odstrani svetlobne odboje.

Slika 6.7 kaže sliko pred in po odstranitvi svetlobnih odbojev leče ter sliko samih svetlobnih odbojev.



Slika 6.2: Serija 8 NDR-slik zgoraj in 2 VDR-sliki spodaj, vsaka generirana z drugo vrsto uteži. Sredina, uteži predlagane s strani Debevca in Malika, spodaj široka funkcija klobuka. Razpon časa izpostavitev od $156\mu s$ do $20000\mu s$. Za preslikavo bravnih tonov je uporabljen Fattal algoritem [4].

Slika 6.3: Serija 8 NDR-slik zgoraj in VDR-slika spodaj. Razpon časa izpostavitev od $107\mu s$ do $13600\mu s$. Za preslikavo bravnih tonov je uporabljen Fattal algoritem.

Slika 6.4: Serija 8 NDR-slik zgoraj in VDR-slika spodaj. Razpon časa izpostavitev od 169μs do 21600μs. Za preslikavo bravnih tonov je uporabljen Fattal algoritem.

Slika 6.5: Serija 8 NDR-slik zgoraj in VDR-slika spodaj. Razpon časa izpostavitev od $187\mu s$ do $24000\mu s$. Za preslikavo bravnih tonov je uporabljen Fattal algoritem.

Slika 6.6: Serija 8 NDR-slik zgoraj in VDR-slika spodaj. Razpon časa izpostavitev od 107μs do 13600μs. Za preslikavo bravnih tonov je uporabljen Fattal algoritem.

Slika 6.7: Slika pred odstranitvijo odbojev zgoraj, po odstranitvi sredina in slika svetlobnih odbojev spodaj.

Poglavje 7

Sklepne ugotovitve

Visoki dinamični razpon je eden od ključnih preskokov, ki se dogaja in se bo dogodil na področju digitalne fotografije.

Današnje fotografije z nizkim dinamičnim razponom ne zadovoljujejo več uporabnikov. Še posebej to velja za strokovnjake in poklicne fotografe. Obstaja namreč veliko prizorov, ko je 256 vrednosti na barvni kanal, kot jih imajo povečini današnji formati fotografij, bistveno premalo, da bi slika zajela celoten razpon svetlobnega sevanja in s tem celotni obseg informacij.

Slike visokega dinamičnega razpona rešujejo prav ta problem. Z drugačnimi pristopi pri zajemanju, shranjevanju in prikazovanju le-teh se vzpostavlja popolnoma nov svet digitalne fotografije, kjer ne bo več potrebno razmišljati, kakšno časovno izpostavitev uporabiti, da se zajame najpomembnejši del prizora. Slike visokega dinamičnega razpona omogočajo prikaz veliko višjega razpona svetlobnega sevanja prizora in tako omogočijo zajetje informacij od najbolj temnih do najbolj svetlih delov slike.

Namen tega diplomskega dela je bil preučiti področje zajemanja slik visokega dinamičnega razpona in probleme, ki se na tem področju pojavljajo, podati rešitve in jih ovrednostiti. Še posebno podrobno pa raziskati zajemanje slik visokega dinamičnega razpona z večkratno izpostavitvijo, kjer so bile, izmed navedenih tehnik v tem delu, izbrane tehnike zajemanja tudi implementirane in praktično preizkušene.

Pregled področja zajemanja slik nakazuje, da obstaja veliko pristopov k reševanju tega problema, da se veliko strokovnjakov z njim ukvarja in da je na trgu že kar nekaj rešitev, katerih edina ovira za uveljavitev so še vedno nekoliko previsoki stroški izdelave. Temu, da se slike visokega dinamičnega razpona še niso razširile v splošni uporabi, lahko dodamo razlog, da še ne obstajajo dovolj dobri prikazovalniki, ki bi zmogli prikazati zajete slike v vsej

njihovi kakovosti. Toda tudi področje prikazovanja sik visokega dinamičnega razpona hitro napreduje, tako da je prehod s starih na nove slike, slike visokega dinamičnega razpona, neizogiben.

Rezultati, slike, ki so bile generirane s pomočjo implementacije algoritmov, podanih v tem delu, kažejo, da so sami postopki zajemanja slik že dokaj dovršeni. Generirane slik so kakovostne in na pogled brez očitnih napak. Le pri izredno kontrastnih in temačnih predelih slik se v večji meri pojavlja na slikah šum, ki bi ga bilo potrebno dodatno odstraniti oziroma izboljšati algoritme.

To diplomsko delo podaja pregled nad zajemanjem slik visokega dinamičnega razpona, še posebno nad zajemanjem z večkratno izpostavitvijo, vendar seveda ne vsebuje vseh informacij s tega področja. Kot nadaljevanje tega dela bi bilo zanimivo raziskati in implementirati tudi druge algoritme s področja zajemanja slik visokega dinamičnega razpona z večkratno izpostavitvijo, še posebno algoritme za odstranjevanje napak zaradi gibanja in algoritme za poravnavo slik, ki v tem delu niso preizkušene. Tako bi lahko ovrednotili tude druge dele tega področja. Ravno tako bi bilo interesantno primerjati različne načine zajemanja slik visokega dinamičnega razpona in tako ugotoviti kakovostne razlike med njimi.

Slike visokega dinamičnega razpona so prihodnost digitalne fotografije, kar dokazuje tudi to diplomsko delo, ki raziskuje probleme in nakazuje rešitve enega izmed najpomembnejših problemov na tem področju.

Dodatek A

Izvorna koda implementacije

genHDRppm.h

```
#include <pam.h>
#include </usr/include/gsl/gsl_matrix.h>

#define R 0
#define G 1
#define B 2

#define X 0
#define Y 1

#define PRINT_RESPONSE_FUNCTION 1

void read_images(char** argv);

gsl_vector *derive_camera_respond_function(int color, double lambda);
pixel **lens_flare_removal(pixel ** hdr, int output);
pixel **merge_to_hdr(gsl_vector *func_r, gsl_vector *func_g, gsl_vector *
    func_b, int w_sel);

void write_image(char *out, pixel **hdr);
```

genHDRppm.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <pam.h>
#include </usr/include/gsl/gsl_matrix.h>
#include </usr/include/gsl/gsl_math.h>

#include "genHDRppm.h"

int num_i;          // number of input_images
pixel ***images;    // array of pointers to input images
int col;            // number of colums
int row;            // number of rows
pixval maxval;      // max pixel value
double *t;          // array of exposure times
```

```
int main( int argc, char** argv)
{
  int i;              // counter
  double lambda;      // weight
  int merge_w_sel;    // weight select
  int flare_removal;  // flare removal select
  int flare_output;   // flare output select

  /* check input parameters */
  if (argc < 10)
  {
    printf( "\nUsage: num_of_input_images input_image1.ppm exp_time1[us]
        input_image2.ppm exp_time2[us] [input_image3.ppm exp_time3...] lambda
        w_sel flare_removal flare_output output_image.ppm\n\n" );
    exit(0);
  }

  /* read number of pictures */
  num_i = atoi(argv[1]);

  /* recheck input parameters */
  if (argc != (num_i * 2 + 7))
  {
    printf( "\nUsage: num_of_input_images input_image1.ppm exp_time1[us]
        input_image2.ppm exp_time2[us] [input_image3.ppm exp_time3...] lambda
        w_sel flare_removal flare_output output_image.ppm\n\n" );
    exit(0);
  }

  /* image allocation */
  images = malloc(num_i * sizeof(pixel *));
  t = malloc(num_i * sizeof(double));
  if (images == NULL)
  {
    puts("\nError: Failure to allocate room for image pointers.\n");
    exit(0);
  }

  /* read input parameters */
  lambda = atof(argv[num_i*2+2]);
  merge_w_sel = atoi(argv[num_i*2+3]);
  flare_removal = atoi(argv[num_i*2+4]);
  flare_output = atoi(argv[num_i*2+5]);

  /* start hdr generation */
  printf( "\nStarting HDR image generator...\n\n" );
  /* read input images */
  read_images(argv);

  /* deriving camera respon function */
  printf("\nDeriving camera respond function (color R)...\n");
  gsl_vector *camera_func_r = derive_camera_respond_function(R, lambda);

  printf("\nDeriving camera respond function (color B)...\n");
  gsl_vector *camera_func_g = derive_camera_respond_function(G, lambda);

  printf("\nDeriving camera respond function (color G)...\n");
  gsl_vector *camera_func_b = derive_camera_respond_function(B, lambda);
```

```
  /* if enabled print camera respond function into file camera_respond_func.
      txt */
#if PRINT_RESPONSE_FUNCTION
  FILE *fp;

  if ((fp = fopen("camera_respond_func.txt", "wt")) == NULL)
  {
    printf("Error: Can not open file camera_respond_func.txt\n");
    exit(0);
  }

  fprintf(fp, "CAMERA RESPOSE FUNCTION\n");
  for (i=0; i<=maxval; i++)
  {
    fprintf(fp, "%f ", gsl_vector_get(camera_func_r, i));
    fprintf(fp, "%f ", gsl_vector_get(camera_func_g, i));
    fprintf(fp, "%f %i\n", gsl_vector_get(camera_func_b, i), i);
  }
  fprintf(fp, "END OF CAMERA RESPONSE FUNCTION");
  fclose(fp);
#endif

  /* merge to HDR */
  printf ("\nMerging to HDR...\n");
  pixel ** hdr_image = merge_to_hdr(camera_func_r, camera_func_g,
      camera_func_b, merge_w_sel);

  /* free input picture arrays */
  for (i=0; i<num_i; i++)
    ppm_freearray(images[i],  row);

  /* If flare removal enabled remove flare */
  if (flare_removal==1)
  {
    printf ("\nFlare removal...\n");
    hdr_image = lens_flare_removal(hdr_image, flare_output);
  }

  /* write output image */
  printf("\nWriting output image... \n");
  write_image(argv[argc-1], hdr_image);

  printf( "\nDone!\n\n" );
  return 0;
}

/* read input images */
void read_images(char** argv)
{
  int i;              // counter
  FILE *fp;           // File pointer
  int col_in, row_in; // temp col/row
  pixval maxval_in;   // temp maxval


  printf("\nReading input images...\n");

  // read all input images
```

```
  for (i=0; i<num_i; i++)
  {
    if ((fp = fopen(argv[(i*2)+2], "rb")) == NULL)
    {
      printf("Error: Can not open file %s\n", argv[(i*2)+2]);
      exit(0);
    }

    images[i] = ppm_readppm(fp, &col_in, &row_in, &maxval_in);
    fclose(fp);

    // read exposure time
    t[i] = log((double)atoi(argv[(i*2)+3]))/log(2) -  log(1000000)/log(2);

    printf("Input image number %d: Widht: %d Length: %d ExpTime: %f\n", i+1,
        col_in, row_in, t[i]);

    // check if same size
     if (i==0)
    {
      col = col_in;
      row = row_in;
      maxval = maxval_in;
    }
    else if ((col_in != col) && (row_in != row) && (maxval_in != maxval))
    {
      printf("Error: Input image size or maxval differ!\n");
      exit(0);
    }
 }
}

/* derive camera respond fuction from input images */
gsl_vector* derive_camera_respond_function(int color, double lambda)
{

  int i,j; //counters
  int Zij; //input pixels

  int num=maxval+1; // number of values

  int sqrt_num_of_pix=sqrt(num*2/num_i)+1; //number of pixels
  int num_of_pix=sqrt_num_of_pix*sqrt_num_of_pix;

  // input pixels
  int Zi[num_of_pix][2];
  for(i=0; i<sqrt_num_of_pix; i++)
  {
    for(j=0; j<sqrt_num_of_pix; j++)
    {
      Zi[i*sqrt_num_of_pix+j][X] = col/sqrt_num_of_pix * i + col/(
          sqrt_num_of_pix*2);
      Zi[i*sqrt_num_of_pix+j][Y] = row/sqrt_num_of_pix * j + row/(
          sqrt_num_of_pix*2);
    }
  }

  // weights
  int w[maxval+1];
```

```
for (i=0; i<=(maxval+1)/2; i++)
  w[i] = i;
for (i=(maxval+1)/2+1; i<=maxval; i++)
  w[i] = maxval - i;

// matrices for SVD calculation
gsl_matrix *A = gsl_matrix_calloc (num_of_pix*num_i+num+1, num+num_of_pix);
gsl_vector *b = gsl_vector_calloc (num_of_pix*num_i+num+1);

int k = 0;
for (i=0;i<num_of_pix;i++)
{
  for (j=0; j<num_i; j++)
  {
      switch (color)
      {
        case R:
          Zij = PPM_GETR(images[j][Zi[i][Y]][Zi[i][X]]);
          break;
        case B:
          Zij = PPM_GETB(images[j][Zi[i][Y]][Zi[i][X]]);
          break;
        default:
          Zij = PPM_GETG(images[j][Zi[i][Y]][Zi[i][X]]);
      }
      gsl_matrix_set (A, k, Zij, (double)w[Zij]);
      gsl_matrix_set (A, k, num+i, (double)-w[Zij]);
      gsl_vector_set (b, k,  (double)(w[Zij] * t[j]));
      k=k+1;
  }
}

// Fix the curve by setting its middle value to 0
gsl_matrix_set (A, k, num/2, 1);
k=k+1;

// Include the smoothness equations
for (i=0; i<num-2; i++)
{
  gsl_matrix_set (A, k, i,   lambda * w[i+1]);
  gsl_matrix_set (A, k, i+1, -2 * lambda * w[i+1]);
  gsl_matrix_set (A, k, i+2, lambda * w[i+1]);
  k=k+1;
}

// Solve the system using SVD
gsl_matrix *V  = gsl_matrix_alloc(num+num_of_pix, num+num_of_pix);
gsl_vector *S  = gsl_vector_alloc(num+num_of_pix);
gsl_vector *work = gsl_vector_calloc(num+num_of_pix);
gsl_vector *func = gsl_vector_calloc(num+num_of_pix);

int ok = gsl_linalg_SV_decomp (A, V, S, work);

ok = gsl_linalg_SV_solve (A, V, S, b, func);

// free unused matrices
gsl_matrix_free(A);
gsl_matrix_free(V);
gsl_vector_free(S);
```

```
  gsl_vector_free(work);
  gsl_vector_free(b);

  // return function
  return func;
}

/* merge input pictures to one HDR picture */
pixel **merge_to_hdr(gsl_vector *func_r, gsl_vector *func_g, gsl_vector *
    func_b, int w_sel)
{
  int i,j,z;                          // counters

  double pixs[num_i][3];              // used pixels
  double w_sum_r, w_sum_g, w_sum_b; // weights
  double hdr_r, hdr_g, hdr_b;         // hdr pixel

  pixel **hdr = ppm_allocarray(col, row); //array for hdr picture

  /* weights generation */
  double w[maxval+1];
  if (w_sel == 0)
  {
    for (i=0; i<=maxval; i++) {
      w[i] = 1 - pow(2*((double)i/maxval)-1, 12);
    }
  }
  else
  {
    for (i=0; i<=(maxval+1)/2; i++)
      w[i] = i;
    for (i=(maxval+1)/2+1; i<=maxval; i++)
      w[i] = maxval - i;
  }

  /* pixel calculation */
  for (j=0; j<row; j++)
  {
    for (i=0; i<col; i++)
    {
      w_sum_r = w_sum_g = w_sum_b = 0;
      hdr_r = hdr_g = hdr_b = 0;

      for (z=0; z<num_i; z++)
      {
        pixs[z][R] = gsl_vector_get(func_r, PPM_GETR(images[z][j][i]));
        pixs[z][G] = gsl_vector_get(func_g, PPM_GETG(images[z][j][i]));
        pixs[z][B] = gsl_vector_get(func_b, PPM_GETB(images[z][j][i]));

        // normalization
        pixs[z][R] = pixs[z][R] - t[z];
        pixs[z][G] = pixs[z][G] - t[z];
        pixs[z][B] = pixs[z][B] - t[z];

        // weights
        hdr_r = hdr_r + pixs[z][R]*w[PPM_GETR(images[z][j][i])];
        hdr_g = hdr_g + pixs[z][G]*w[PPM_GETG(images[z][j][i])];
        hdr_b = hdr_b + pixs[z][B]*w[PPM_GETB(images[z][j][i])];
```

```
        // sum of weights
        w_sum_r = w_sum_r + w[PPM_GETR(images[z][j][i])];
        w_sum_g = w_sum_g + w[PPM_GETG(images[z][j][i])];
        w_sum_b = w_sum_b + w[PPM_GETB(images[z][j][i])];
      }
        hdr_r = hdr_r/w_sum_r;
        hdr_g = hdr_g/w_sum_g;
        hdr_b = hdr_b/w_sum_b;

        // save pixels in linear space
        PPM_ASSIGN(hdr[j][i], floor(pow(2,hdr_r)+0.5), floor(pow(2,hdr_g)+0.5)
            , floor(pow(2,hdr_b)+0.5));

        // output cliping
        if(PPM_GETR(hdr[j][i]) > 65535)
          PPM_ASSIGN(hdr[j][i], 65535, PPM_GETG(hdr[j][i]), PPM_GETB(hdr[j][i
              ]));
        if(PPM_GETG(hdr[j][i]) > 65535)
          PPM_ASSIGN(hdr[j][i], PPM_GETR(hdr[j][i]), 65535, PPM_GETB(hdr[j][i
              ]));
        if(PPM_GETB(hdr[j][i]) > 65535)
          PPM_ASSIGN(hdr[j][i], PPM_GETR(hdr[j][i]), PPM_GETG(hdr[j][i]),
              65535);

        // save maxval
              if (PPM_GETR(hdr[j][i])>maxval)
                maxval = PPM_GETR(hdr[j][i]);
              if (PPM_GETG(hdr[j][i])>maxval)
                maxval = PPM_GETG(hdr[j][i]);
              if (PPM_GETB(hdr[j][i])>maxval)
                maxval = PPM_GETB(hdr[j][i]);
    }
  }

  // retun hdr image
  return hdr;

}

/* Lens removal */
pixel **lens_flare_removal(pixel **hdr, int output)
{
  int i,j,k,l;            // counters

  int minpix=1000;        // min pixel
  int num_hot = 0;        // number of hot pixels

  int reduce = col/128;   // reduced ratio
  int row_r = row/reduce; // reduced rows
  int col_r = col/reduce; // reduced colums

  // reduced images
  pixel **icr = ppm_allocarray(col_r, row_r);
  pixel **igr = ppm_allocarray(col_r, row_r);

  // initialize image
  for (j=0; j<row_r; j++)
  {
    for (i=0; i<col_r; i++)
```

```
  {
    PPM_ASSIGN(icr[j][i], 0, 0, 0);
  }
}

// downsample image
for (j=0; j<row; j++)
{
  for (i=0; i<col; i++)
  {
    PPM_ASSIGN(icr[j/reduce][i/reduce], PPM_GETR(hdr[j][i]) + PPM_GETR(icr[j
        /reduce][i/reduce]),
                                        PPM_GETG(hdr[j][i]) + PPM_GETG(icr[j
                                            /reduce][i/reduce]),
                                        PPM_GETB(hdr[j][i]) + PPM_GETB(icr[j
                                            /reduce][i/reduce]));
  }
}

// make grayscale image
for (j=0; j<row_r; j++)
{
  for (i=0; i<col_r; i++)
  {
    PPM_ASSIGN(icr[j][i], PPM_GETR(icr[j][i])/(reduce*reduce),
                          PPM_GETG(icr[j][i])/(reduce*reduce),
                          PPM_GETB(icr[j][i])/(reduce*reduce));

    PPM_ASSIGN(igr[j][i], 0, (54*PPM_GETR(icr[j][i]) + 183*PPM_GETG(icr[j][i
        ]) + 19*PPM_GETB(icr[j][i]))/256, 0);

    // save min pixel
    if (PPM_GETG(igr[j][i])<minpix)
    {
      minpix=PPM_GETG(igr[j][i]);
    }
  }
}

if(minpix == 0)
  minpix=1;

// check hot pixels
for (j=0; j<row_r; j++)
{
  for (i=0; i<col_r; i++)
  {
    if (PPM_GETG(igr[j][i])>1000*minpix)
    {
      PPM_ASSIGN(igr[j][i], 1, PPM_GETG(igr[j][i]), 0);
      num_hot++;
    }
  }
}

//printf("num hot %d\n", num_hot);
// list hot pixels
int hot[num_hot][2];
k=0;
```

```
for (j=0; j<row_r; j++)
{
  for (i=0; i<col_r; i++)
  {
    if (PPM_GETR(igr[j][i]) == 1)
    {
      hot[k][X] = i;
      hot[k][Y] = j;
      k++;
    }
  }
}

/* get minimal ratios for  PSF */
int dist;                              // distance
double dst;                            // distance
int tmp_hot;                           // temp hot pixels value
int start_range = 3;                   // start distance for calculations
int range = 32;                        // end distance for calculations
double tmp_ratio = 0;                  // temp ratio value
double min_ratio[range];               // array of minimal ratios
int min_ratio_inx[range][2];           // array of minimal ratio indexes

min_ratio[start_range-2] = 1;          // initialize
min_ratio_inx[start_range-2][Y] = 0;
min_ratio_inx[start_range-2][X] = 0;
for (k=start_range; k<=range; k++)
{
  min_ratio_inx[k-1][X] = min_ratio_inx[k-2][X];
  min_ratio_inx[k-1][Y] = min_ratio_inx[k-2][Y];
  min_ratio[k-1] = min_ratio[k-2];
  for (j=0; j<row_r; j++)
  {
    for (i=0; i<col_r; i++)
    {
      if(PPM_GETR(igr[j][i]) == 0)
      {
        tmp_hot=0;

        for(l=0; l<num_hot; l++)
        {
          dist = floor(sqrt(pow(hot[l][X] - i, 2) + pow(hot[l][Y] - j, 2)) +
              0.5);
          if(dist == k)
          {
            tmp_hot = tmp_hot + PPM_GETG(igr[hot[l][Y]][hot[l][X]]);
          }
        }

        if(tmp_hot > 0)
        {
          tmp_ratio = (double)PPM_GETG(igr[j][i])/tmp_hot;

          if(tmp_ratio < min_ratio[k-1])
          {
            min_ratio_inx[k-1][Y] = j;
            min_ratio_inx[k-1][X] = i;
            min_ratio[k-1] = tmp_ratio;
          }
```

```
        }
      }
    }
  }
}

// calculate PFS
gsl_vector *Pi = gsl_vector_calloc (range-start_range+1);
gsl_matrix *Pj = gsl_matrix_calloc (range-start_range+1, 4);

for (k=start_range; k<=start_range; k++)
{
  gsl_vector_set(Pi, k-start_range, PPM_GETG(igr[min_ratio_inx[k-1][Y]][
      min_ratio_inx[k-1][X]]));

  for (l=0; l<num_hot; l++)
  {
    dst = sqrt(pow(hot[l][X] - min_ratio_inx[k-1][X], 2) + pow(hot[l][Y] -
        min_ratio_inx[k-1][Y], 2));
    if (dst < 3)
      dst = 3.0;
    gsl_matrix_set(Pj, k-start_range, 0, gsl_matrix_get(Pj, k-start_range,
        0) + (double)PPM_GETG(igr[hot[l][Y]][hot[l][X]]));
    gsl_matrix_set(Pj, k-start_range, 1, gsl_matrix_get(Pj, k-start_range,
        1) + (double)PPM_GETG(igr[hot[l][Y]][hot[l][X]])/dst);
    gsl_matrix_set(Pj, k-start_range, 2, gsl_matrix_get(Pj, k-start_range,
        2) + (double)PPM_GETG(igr[hot[l][Y]][hot[l][X]])/(dst*dst));
    gsl_matrix_set(Pj, k-start_range, 3, gsl_matrix_get(Pj, k-start_range,
        3) + (double)PPM_GETG(igr[hot[l][Y]][hot[l][X]])/(dst*dst*dst));

  }
}

gsl_matrix *V  = gsl_matrix_alloc(4, 4);
gsl_vector *S  = gsl_vector_alloc(4);
gsl_vector *work = gsl_vector_calloc(4);
gsl_vector *psf_c = gsl_vector_calloc(4);

int ok = gsl_linalg_SV_decomp (Pj, V, S, work);

ok = gsl_linalg_SV_solve (Pj, V, S, Pi, psf_c);

// free unused arrays
gsl_vector_free(Pi);
gsl_matrix_free(Pj);
gsl_matrix_free(V);
gsl_vector_free(S);
gsl_vector_free(work);

// calculate flare
double pfs;
gsl_matrix *fcr_r  = gsl_matrix_calloc(col_r, row_r);
gsl_matrix *fcr_g  = gsl_matrix_calloc(col_r, row_r);
gsl_matrix *fcr_b  = gsl_matrix_calloc(col_r, row_r);
for (j=0; j<row_r; j++)
{
  for (i=0; i<col_r; i++)
  {
    for (k=0; k<num_hot; k++)
```

```
    {
      dst = sqrt(pow(hot[k][X] - i, 2) + pow(hot[k][Y] - j, 2));
        if (dst<3)
          dst = 3;

        pfs = gsl_vector_get(psf_c, 0) + gsl_vector_get(psf_c, 1)/dst +
            gsl_vector_get(psf_c, 2)/(dst*dst) + gsl_vector_get(psf_c, 3)/(
            dst*dst*dst);

        gsl_matrix_set(fcr_r, i, j, gsl_matrix_get(fcr_r, i, j) + (double)
            PPM_GETR(icr[j][i])*pfs);
        gsl_matrix_set(fcr_g, i, j, gsl_matrix_get(fcr_g, i, j) + (double)
            PPM_GETG(icr[j][i])*pfs);
        gsl_matrix_set(fcr_b, i, j, gsl_matrix_get(fcr_b, i, j) + (double)
            PPM_GETB(icr[j][i])*pfs);
    }
  }
}

// check flare size
double max_ratio = 1;
for (j=0; j<row_r; j++)
{
  for (i=0; i<col_r; i++)
  {
      tmp_ratio = gsl_matrix_get(fcr_r, i, j) / (double)PPM_GETR(icr[j][i]);
      if (tmp_ratio > max_ratio)
        max_ratio = tmp_ratio;

      tmp_ratio = gsl_matrix_get(fcr_g, i, j) / (double)PPM_GETG(icr[j][i]);
      if (tmp_ratio > max_ratio)
        max_ratio = tmp_ratio;

      tmp_ratio = gsl_matrix_get(fcr_b, i, j) / (double)PPM_GETB(icr[j][i]);
      if (tmp_ratio > max_ratio)
        max_ratio = tmp_ratio;
  }
}

// correct flare if size to big
if (max_ratio > 1)
{
  for (j=0; j<row_r; j++)
  {
    for (i=0; i<col_r; i++)
    {
        gsl_matrix_set(fcr_r, i, j, gsl_matrix_get(fcr_r, i, j)/max_ratio);
        gsl_matrix_set(fcr_g, i, j, gsl_matrix_get(fcr_g, i, j)/max_ratio);
        gsl_matrix_set(fcr_b, i, j, gsl_matrix_get(fcr_b, i, j)/max_ratio);
    }
  }
}

// interpolate flare image and substract from original image
double x1, x2, y1, y2;
double int_r1, int_r2, int_r, int_g1, int_g2, int_g, int_b1, int_b2, int_b;
int offset=reduce/2;

for (j=0; j<row; j++)
```

```
  {
    for (i=0; i<col; i++)
    {

      if ((j<offset) || (j>=row-offset) || (i<offset) ||  (i>=col-offset))
      {
        int n = 1;
      }
      else
      {
        x1 = ((i-offset)/reduce)*reduce + offset;
        x2 = ((i-offset)/reduce+1)*reduce + offset - 1;
        y1 = ((j-offset)/reduce)*reduce + offset;
        y2 = ((j-offset)/reduce+1)*reduce + offset - 1;

        int_r1 = (x2 - i)/reduce * gsl_matrix_get(fcr_r, (i-offset)/reduce,
            (j-offset)/reduce)   + (i - x1)/reduce * gsl_matrix_get(fcr_r, (i-
            offset)/reduce+1, (j-offset)/reduce);
        int_r2 = (x2 - i)/reduce * gsl_matrix_get(fcr_r, (i-offset)/reduce,
            (j-offset)/reduce+1) + (i - x1)/reduce * gsl_matrix_get(fcr_r, (i-
            offset)/reduce+1, (j-offset)/reduce+1);
        int_r  = (y2 - j)/reduce * int_r1 + (j - y1)/reduce * int_r2;

        int_g1 = (x2 - i)/reduce * gsl_matrix_get(fcr_g, (i-offset)/reduce,
            (j-offset)/reduce)   + (i - x1)/reduce * gsl_matrix_get(fcr_g, (i-
            offset)/reduce+1, (j-offset)/reduce);
        int_g2 = (x2 - i)/reduce * gsl_matrix_get(fcr_g, (i-offset)/reduce,
            (j-offset)/reduce+1) + (i - x1)/reduce * gsl_matrix_get(fcr_g, (i-
            offset)/reduce+1, (j-offset)/reduce+1);
        int_g  = (y2 - j)/reduce * int_g1 + (j - y1)/reduce * int_g2;

        int_b1 = (x2 - i)/reduce * gsl_matrix_get(fcr_b, (i-offset)/reduce,
            (j-offset)/reduce)   + (i - x1)/reduce * gsl_matrix_get(fcr_b, (i-
            offset)/reduce+1, (j-offset)/reduce);
        int_b2 = (x2 - i)/reduce * gsl_matrix_get(fcr_b, (i-offset)/reduce,
            (j-offset)/reduce+1) + (i - x1)/reduce * gsl_matrix_get(fcr_b, (i-
            offset)/reduce+1, (j-offset)/reduce+1);
        int_b  = (y2 - j)/reduce * int_b1 + (j - y1)/reduce * int_b2;

            if (output != 1)
      {
        int_r = (double)PPM_GETR(hdr[j][i]) - int_r + 0.5;
        int_g = (double)PPM_GETG(hdr[j][i]) - int_g + 0.5;
        int_b = (double)PPM_GETB(hdr[j][i]) - int_b + 0.5;
      }

        if(int_r < 0) int_r = 0;
        if(int_g < 0) int_g = 0;
        if(int_b < 0) int_b = 0;

        PPM_ASSIGN(hdr[j][i], (int)int_r, (int)int_g, (int)int_b);

      }
    }
  }

  // return flare free image
  return hdr;
}
```

```
/* write output image to file */
void write_image(char *out, pixel **hdr)
{
  FILE *fp;
  int i,j;


  if ((fp = fopen(out, "w")) == NULL)
  {
    printf("Error: Can not open file %s\n", out);
    exit(0);
  }
  ppm_writeppm(fp, hdr, col, row, maxval, 0);
  fclose(fp);
}
```

Slike

Tabele

Literatura

[1] M. Aggarwal, N. Ahuja, "Split Aperture Imaging for High Dynamic Range," v zborniku *Eighth IEEE International Conference on Computer Vision,* Vancouver, Kanada, julij 2001, zv. 2, str. 10-17.

[2] V. Brajovic, T. Kanade, "A sorting image sensor: An example of massively parallel intensity-to-time processing for low-latency computational sensors," v zborniku *IEEE International Conference on Robotics and Automation,* Minneapolis, ZDA, april 1996, zv. 2, str. 1638–1643.

[3] P. E. Debevec, J. Malik, "Recovering High Dynamic Range Radiance Maps from Photographs," v zborniku *Proceedings of SIGGRAPH 97,* Computer Graphics Proceedings, Annual Conference Series, avgust 1997, str. 369-378.

[4] R. Fattal, D. Lischinski in M. Werman, "Gradient domain high dynamic range compression," v zborniku *29th annual conference on Computer graphics and interactive techniques,* San Antonio, ZDA, 2002, str. 249-256.

[5] J. A. Ferwerda, "Elements of early vision for computer graphics," *IEEE Computer Graphics and Applications,* juli-avgust 2001, zv. 21, št. 5, str. 22-33.

[6] K. Jacobs, G. Ward, in C. Loscos, "Automatic hdri generation of dynamic scenes" *IEEE Computer Graphics and Applications*, marec-april 2008, zv. 28, št. 2, str. 84-93.

[7] E. A. Khan, A. O. Akyuz in E. Reinhard, "Ghost removal in high dynamic range images", v zborniku *IEEE International Conference on Image Processing,* Atlanta, ZDA, oktober 2006, str. 2005-2008.

[8] T. Mitsunaga, S. K. Nayar, "Radiometric Self Calibration" v zborniku *IEEE Computer Society Conference on Computer Vision and Pattern Recognition,* Fort Collins, ZDA, junij 1999, zv. 1, str. 374-380.

[9] S. K. Nayar, T. Misunaga, "High dynamic range imaging: Spatially varying pixel exposures," v zborniku *IEEE Conference on Computer Vision and Pattern Recognition,* Hilton Head Island, ZDA, junij 2000, zv. 1, str. 472-479.

[10] E. Reinhard, G. Ward, S. Pattanaik in P. Debevec, *High Dynamic Range Imaging: Acquisition, Display, and Image-Based Lighting*, San Francisco: Morgan Kaufman, 2006, pogl. 1, 2, 3, 4, 5, 6.

[11] R. A. Street, *High dynamic range segmented pixel sensor array*, U.S. Patent 5789737, avgust 1998, dostopno na: http://www.patentstorm.us/patents/5789737/description.html.

[12] *http://www.cvrl.org.*

[13] *http://www.dolby.com/promo/hdr/technology.html.*

[14] *http://en.wikipedia.org/wiki/Color_ depth.*

[15] *http://netpbm.sourceforge.net/.*

[16] *ftp://ftp.gnu.org/gnu/gsl/gsl-1.11.tar.gz.*

[17] *http://downloads.sourceforge.net/qtpfsgui/qtpfsgui-1.8.12.tar.gz.*

[18] *http://scanline.ca/exrtools/exrtools-0.4.tar.gz.*

Izjava

Izjavljam, da sem diplomsko nalogo izdelal samostojno pod vodstvom mentorja prof. dr. Aleša Leonardisa. Izkazano pomoč drugih sodelavcev sem v celoti navedel v zahvali.

Ljubljana, datum Peter Kragelj