UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika – 2. stopnja

Katarina Zadražnik

Likovno upodabljanje slik

Magistrsko delo

Mentor: izred. prof. dr. Andrej Bauer

Ljubljana, 2014

Podpisana Katarina Zadražnik izjavljam:

– da sem magistrsko delo z naslovom ?????????????????? izdelala samostojno pod mentorstvom izred. prof. dr. Andreja Bauerja, ter

– da Fakulteti za matematiko in fiziko Univerze v Ljubljani dovoljujem objavo elektronske oblike svojega dela na spletnih straneh.

Ljubljana, 10. oktober 2014 Podpis:

Zahvala

Tralala

Kazalo

Program dela **ix**

Povzetek **xi**

1 Uvod **1**
1.1 Pravzaprav uvod . . . 1
1.2 Kažipot . . . 2
1.3 Programerski del . . . 3

2 Obdelava slik **5**
2.1 Fourierova transformacija . . . 5
2.1.1 Fourierova baza . . . 6
2.1.2 Dvodimenzionalna diskretna Fourierova transformacija . . . 8
2.1.3 Translacijsko invariantne linearne transformacije . . . 10
2.1.4 Hitra Fourierova transformacija . . . 15
2.2 Transformacije slik . . . 21
2.2.1 Signali . . . 21
2.2.2 Točkovne transformacije . . . 21
2.2.3 Lokalne transformacije . . . 21
2.2.4 Globalne transformacije . . . 25
2.3 Barvni modeli . . . 27
2.3.1 RGB in CMY barvna modela . . . 27
2.3.2 HSV in HSL barvna modela . . . 28
2.3.3 Kubelka–Monk barvni model . . . 28
2.4 Odstranjevanje šuma na sliki . . . 28
2.4.1 Metoda šuma . . . 28
2.4.2 Lokalni algoritmi za odstranjevanje šuma na sliki . . . 29
2.4.3 Nelokalni algoritem za odstranjevanje šuma na sliki . . . 31
2.4.4 Implementacija in primeri . . . 34

3 Generiranje šuma in tekstura papirja **35**

4 Likovno upodabljanje slik **37**
4.1 Artistični filter . . . 37
4.2 Splošni pregled dela na področju LUS . . . 37
4.2.1 Ilustracije . . . 37
4.2.2 Risbe . . . 38

4.2.3 Painterly rendering 39

5 Likovno upodabljanje s potezami 41
5.1 Optimizacijski algoritmi 42
5.1.1 Voronojevi algoritmi 43
5.1.2 Izkustveni algoritmi 46
5.2 Požrešne metode 48
5.2.1 Slikanje z ravnimi potezami 49
5.2.2 Slikanje z Beizerovimi krivuljami 51
5.3 Anizotropični čopič za likovno upodabljanje 54
5.3.1 Upodabljanje potez z anizotropičnim čopičem 55
5.3.2 Lighting effect 56
5.3.3 Fast paint texture 57

6 Risanje z voščenkami 59
6.1 Model voščenke in papirja 59
6.1.1 Model voščenke 59
6.1.2 Model papirja 61
6.2 Modeliranje risanja z voščenkami 61
6.2.1 Osnovne količine in algoritem za risanje posameznih linij 61
6.2.2 Prilagoditev profila voščenke 62
6.2.3 Trenje 63
6.3 Smearing 64
6.4 Redepozicija 65
6.5 Upodabljanje z voščenkami 65

7 Risanje s svičnikom 69
7.1 Linijska slika 69
7.1.1 Classification 69
7.1.2 Line shaping 70
7.2 Tone drawing 71
7.2.1 Tone Map Generation 71
7.2.2 Model-based Tone Transfer 71
7.2.3 Parameter learning 72
7.2.4 Pencil Texture Rendering 72
7.3 Color Pencil Drawing 73

Literatura 75

Program dela

Ljubljana, datum

izred. prof. dr. Andrej Bauer

Povzetek

Povzetek bo na koncu.

Abstract

Math. Subj. Class. (MSC 2010): XXXXXX, YYYYYY
Ključne besede: foo, bar, baz
Keywords: foo, bar, baz

Poglavje 1

Uvod

1.1 Pravzaprav uvod

Matematika je veda, ki govori o vzorcih, simetrijah, oblikah in lepoti medsebojnega prepletanja vsega tega. Je čudovit abstrakten svet, ki ne dopušča vsakomur, da bi vstopil vanj. Obiskovalci morajo namreč pri vstopu v ta svet sprejeti pravila, ki so zapisana v obliki definicij, izrekov, lem, trditev, posledic ..., in ravno ta pravila preplašijo grupe posameznikov, ki v prihodnje morda včasih zatavajo le še v ε-okolico matematičnega sveta (pri čemer ε ni nujno poljubno majhno število). Pa vendarle obstaja množica ljudi, ki živi v tem svetu, vztraja v njem, ga nenehno raziskuje in pri tem odkriva vedno nove in nove zaklade (bolj kot sam zaklad na koncu, je pravzaprav pomembna pot, ki jo je moral posameznik prehoditi, da je prišel do njega[1]).

Prav posebna lepota tega sveta pa se izraža v sodelovanju z ostalimi svetovi. Naokrog ponuja zaključene teorije, recepte, napotke, algoritme, postopke, račune, navodila ..., ponuja jim svoje znanje. Pa vendarle je ta velikodušnost prevečkrat spregledana. Verjetno zaradi matematičnih skrivnosti, ki se bojijo izstopiti iz svojega in oditi v druge svetove, kjer postanejo neuporabne in jih nihče ne razume. Toda zakaj bi bilo narobe, če nekatere stvari ostanejo skrite pred ostalimi—postanejo same sebi namen—saj so ravno te skrivnosti tiste, ki dajejo matematičnemu svetu dušo in ga ohranjajo pri življenju. Ravno prejle je odšel paket znanja iz dežele Dinamičnih sistemov, naslovljen na svet Biologije. Iz dežele Grafov so poslali paket z nekaj koristnimi informacijami o heksagonalnih grafih v svet Kemije. Pošiljali smo tudi v svetove Medicine, Ekonomije ...

Ne smemo pozabiti seveda na svet *Računalništva in programiranja.* In ko ga že omenjamo, povejmo, da je ravno matematika sredstvo, ki v njihovem svetu omogoča sporazumevanje. Če srečaš koga iz njihovega sveta, te bo ta prav lepo pozdravil: "01000100 01001111 01000010 01000101 01010010 00100000 01000100 01000001 01001110".[2] Prava posebnost tega sveta so drevesa, ki pa so nekolika neobičajna, saj so namreč obrnjena na glavo.[3] V nekaterih deželah sestavljajo računalnike, v drugih se ukvarjajo z osnovami računalništva in teorijo, ki je skrita zadaj za vsem tem (ta ima nekatere tesne sodelavce

[1] Zakladi so matematične trditve, izreki, trditve, nove resnice ... Njihovi dokazi in postopki, ki nas pripeljejo do končnih rezultatov, pa so poti, ki jih moramo prehoditi. Kljub temu, da vemo, da na zemljevidu sveta Matematike obstaja polno zakladov, pa do vseh trenutno še nismo uspeli priti, do nekaterih zakladov pa poti sploh ne obstajajo.

[2] V nekaterih narečjih ta pozdrav zveni kot "68 79 66 69 82 32 68 65 78".

[3] Morda bolje rečeno, da so "obrnjena na krošnjo".

v svetu Matematike). V programerskih deželah se zdi, da je vse le množica ukazov za računalnike, pa vendarle je programiranje veliko več kot le to–je umetnost kreativnega pisanja programov za reševanje problemov.

Ob omembi besede umetnost, se spomnim, da nismo še ničesar povedali o svetu Umetnosti, kamor redno pošiljamo pakete znanja. Namreč *umetnost* je kreativno izražanje, ki se izraža v matematičnih strukturiranih formah. Pa naj gre tu za glasbo, za likovno umetnost, ples ... povsod je na nekem nivoju posredno prisotna matematika. Umetniki jo uporabljajo popolnoma podzavestno, ko ustvarjajo simetrične vzorce, plešejo v ponavljajočih se korakih in ustvarjajo harmonične zvoke na glasbilih. Posamezniki med umetniki matematiko uporabljajo kot sredstvo izražanja svoje kreativnosti in si pri tem pomagajo tudi z računalnikom.

Kot matematiki se sprašujemo o presekih množic. In če se vprašamo kaj leži v preseku svetov Matematike, Računalništva in programiranja ter Umetnosti, je odgovor naslednji: "Nekaj lepega, nekaj kar navdihuje ustvarjanje in nekaj kar je vredno raziskovati."

1.2 Kažipot

V magistrskemu delu se bomo posvetili likovnemu upodabljanju slik. Vhodno sliko (fotografijo) bomo s pomočjo algoritmov pretvorili v mozaik, risbo narisano s svinčnikom ali voščenkami in sliko narisano s čopičem. V ta namen bomo tekom magistrskega dela predstavili različne algoritme. Za predstavo, kaj bodo naši algoritmi zmogli, si poglejmo primer na sliki 1.1.

Slika 1.1: Tu bo neka slikica, ki bo lepa.

1.3 Programerski del

Poglavje 2

Obdelava slik

Pri obdelavi signalov in v teoriji filtrov je Fourierova transformacija še vedno zelo močno orodje, ki se ga lahko uporablja za odstranjevanje šuma, pohitritev računanja konvolucije dveh matrik, izboljšavo kakovosti signala ...

V prvem razdelku tega poglavja si bomo zato najprej pogledali teoretično ozadje za Fourierovo transformacijo in njeno izboljšano različico (hitro Fourierovo transformacijo). V našem delu se bomo ukvarjali le z obdelavo slik, zato bomo v pretežni meri vso teorijo in izpeljane postopke aplicirali na slike, čeprav bo vse skupaj v prilagojenih različicah veljalo tudi za ostale vrste signalov.

V drugem razdelku bomo nato spoznali nekaj osnovnih pojmov in postopkov iz teorije filtrov. Našteli bomo osnovne vrste transformacij slike, se naučili postopkov za njihov izračun in nato predstavili osnovne filtrirne postopke (zaznava robov na sliki, zameglitev slike, glajenje robov na sliki ...), ki jih bomo uporabljali v algoritmih za likovno upodabljanje slik.

V tretjem razdelku se bomo posvetili barvnim prostorom RGB, CMY, HSV in YUV. Spoznali bomo tudi Kubelka-Monk barvni model, ki ga bomo uporabili v algoritmu za likovno upodabljanje slik z voščenkami.

Pri obdelavi slik je postopek odstranjevanja šuma zelo pomemben, zato si bomo v zadnjem razdelku tega poglavja pogledali različne algoritme za odstranjevanje šuma. Gaussovo filtriranje je postopek za odstranjevanje šuma, ki je sicer hiter, vendar pa nam večkrat vrne nezadovljive rezultate. Na drugi strani pa bomo spoznali še nelokalno odstranjevanje šuma s slike, ki da zelo dobre rezultate, vendar pa ima kljub temu, da lahko nekatere izračune v algoritmu pohitrimo, še vedno kvadratno časovno zahtevnost. V algoritmih za likovno upodabljanje slik bomo odvisno od naših trenutnih prioritet (hitrost ali kakovost) izbrali ustrezni postopek za odstranjevanje šuma.

2.1 Fourierova transformacija

V tem razdelku bomo predstavili teorijo, ki nas bo vodila do algoritma za izračun dvodimenzionalne hitre (diskretne) Fourierove transformacije. Teorijo bomo povzeli po [8, 2. poglavje] in njene rezultate skupaj z dokazi zapisali za dvodimenzionalno različico ter jih opremili z lastnimi primeri.

2.1.1 Fourierova baza

Najprej bomo definirali N-dimenzionalni vektorski prostor $\ell^2(\mathbb{Z}_N)$ nad kompleksnimi števili:

$$\ell^2(\mathbb{Z}_N) \coloneqq \{z = (z(0), z(1), \ldots, z(N-1)) \mid z(j) \in \mathbb{C}, 0 \leq j \leq N-1\}\,.$$

Prostor bomo opremili s standardno evklidsko bazo $E = \{e_0, e_1, \ldots, e_{N-1}\}$, kjer za bazni element e_j velja: $e_j(n) = 1$ za $j = n$ in $e_j(n) = 0$ sicer. V nadaljevanju bomo definirali prostor $l^2(\mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2})$, ga opremili s standardno bazo in nato spoznali Fourierovo bazo.

Definicija 2.1 Za naravni števili N_1 in N_2 definiramo

$$\ell^2(\mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2}) \coloneqq \{z : \mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2} \to \mathbb{C}\}\ .$$

Elemente $z \in \ell^2(\mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2})$ lahko zapišemo tudi z matriko

$$z = \begin{bmatrix} z(0,0) & z(0,1) & \ldots & z(0,N_2-1) \\ \vdots & \vdots & \ddots & \vdots \\ z(N_1-1,0) & z(N_1-1,1) & \ldots & z(N_1-1,N_2-1) \end{bmatrix}\,,$$

kjer so za $0 \leq n_1 \leq N_1 - 1, 0 \leq n_2 \leq N_2 - 1$ vrednosti $z(n_1, n_2) \in \mathbb{C}$.

Z običajnim seštevanjem in množenjem s skalarjem po komponentah postane množica $\ell^2(Z_{N_1} \times Z_{N_2})$ vektorski prostor nad $\mathbb{C}$. Za elementa $z, w \in \ell^2(Z_{N_1} \times Z_{N_2})$ definiramo kompleksni skalarni produkt

$$\langle z, w\rangle \coloneqq \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} z(n_1, n_2)\overline{w(n_1, n_2)}\ . \tag{2.1}$$

Trditev 2.2 *Prostor $\ell^2(\mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2})$ je N_1N_2-dimenzionalen.*

Dokaz. Dimenzija prostora je enaka razsežnosti njegove baze. Definiramo množico

$$E = \{e_{i,j} \mid 0 \leq i \leq N_1 - 1, 0 \leq j \leq N_2 - 1\}\,,$$

kjer je $e_{i,j}(n_1, n_2) = 1$ za $(i, j) = (n_1, n_2)$ in $e_{i,j}(n_1, n_2) = 0$ sicer. Zanjo enostavno preverimo, da je linearno neodvisna in razpenja celoten prostor $\ell^2(\mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2})$, torej je N_1N_2-razsežna baza prostora $\ell^2(\mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2})$. ∎

Izrek 2.3 *Naj bosta $\{B_0, B_1, \ldots, B_{N_1-1}\}$ in $\{C_0, C_1, \ldots, C_{N_1-1}\}$ ortonormirani bazi za $\ell^2(\mathbb{Z}_{N_1})$ in $\ell^2(\mathbb{Z}_{N_2})$. Za $0 \leq m_1 \leq N_1 - 1$ in $0 \leq m_2 \leq N_2 - 1$ naj bo*

$$D_{m_1,m_2}(n_1, n_2) \coloneqq B_{m_1}(n_1)C_{m_2}(n_2)\ .$$

Tedaj je $\{D_{m_1,m_2}\}_{0\leq m_1\leq N_1-1, 0\leq m_2\leq N_2-1}$ ortonormirana baza prostora $\ell^2(\mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2})$.

Dokaz. Najprej preverimo, da velja enakost $\langle D_{m_1,m_2}, D_{k_1,k_2}\rangle = \langle B_{m_1}, B_{k_1}\rangle\langle C_{m_2}, C_{k_2}\rangle$. Z uporabo definicije skalarnega produkta (2.1) in s preureditvijo členov v dvojni vsoti dobimo:

$$\begin{aligned}
\langle D_{m_1,m_2}, D_{k_1,k_2}\rangle &= \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} D_{m_1,m_2}(n_1,n_2)\overline{D_{k_1,k_2}(n_1,n_2)} \\
&= \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} B_{m_1}(n_1)C_{m_2}(n_2)\overline{B_{k_1}(n_1)C_{k_2}(n_2)} \\
&= \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} B_{m_1}(n_1)\overline{B_{k_1}(n_1)}C_{m_2}(n_2)\overline{C_{k_2}(n_2)} \\
&= \sum_{n_1=0}^{N_1-1} B_{m_1}(n_1)\overline{B_{k_1}(n_1)}\sum_{n_2=0}^{N_2-1} C_{m_2}(n_2)\overline{C_{k_2}(n_2)} \\
&= \langle B_{m_1}, B_{k_1}\rangle\langle C_{m_2}, C_{k_2}\rangle\ .
\end{aligned}$$

Z računom

$$\langle D_{m_1,m_2}, D_{k_1,k_2}\rangle = \langle B_{m_1}, B_{k_1}\rangle\langle C_{m_2}, C_{k_2}\rangle = \begin{cases} 1 & \text{če } (m_1,m_2)=(k_1,k_2); \\ 0 & \text{sicer} \end{cases}$$

pokažemo, da je baza $\{D_{m_1,m_2}\}_{0\le m_1\le N_1-1, 0\le m_2\le N_2-1}$ res ortonormirana (v računu uporabimo dejstvo, da sta $\{B_{m_1}\}_{0\le m_1\le N_1-1}$ in $\{C_{m_2}\}_{0\le m_2\le N_2-1}$ ortonormirani bazi). ∎

V vektorskem prostoru $\ell^2(\mathbb{Z}_N)$ so ortonormirani bazni elementi za $m,n\in\{0,\dots,N-1\}$ definirani kot $E_m(n) \coloneqq \frac{1}{\sqrt{N}} e^{\frac{2\pi\imath mn}{N}}$. Za $m_1\in\{0,\dots,N_1-1\}$ in $m_2\in\{0,\dots,N_2-1\}$ definiramo elemente $E_{m_1,m_2}\in\ell^2(\mathbb{Z}_{N_1}\times\mathbb{Z}_{N_2})$ s formulo

$$E_{m_1,m_2}(n_1,n_2) \coloneqq \frac{1}{\sqrt{N_1N_2}} e^{\frac{2\pi\imath m_1n_1}{N_1}} e^{\frac{2\pi\imath m_2n_2}{N_2}}\ .$$

Spodnja trditev je direktna posledica zgornjega izreka.

Trditev 2.4 *Množica $\{E_{m_1,m_2}\}_{0\le m_1\le N_1-1, 0\le m_2\le N_2-1}$ je ortonormirana baza vektorskega prostora $\ell^2(\mathbb{Z}_{N_1}\times\mathbb{Z}_{N_2})$.*

Bazne elemente E_{m_1,m_2} pomnožimo z $\frac{1}{\sqrt{N_1N_2}}$ in dobimo nove bazne elemente

$$F_{m_1,m_2}(n_1,n_2) \coloneqq \frac{1}{N_1N_2} e^{\frac{2\pi\imath m_1n_1}{N_1}} e^{\frac{2\pi\imath m_2n_2}{N_2}},$$

ki pa posledično niso več normirani.

Definicija 2.5 Množica $F \coloneqq \{F_{m_1,m_2}\}_{0\le m_1\le N_1-1, 0\le m_2\le N_2-1}$ je *Fourierova baza* prostora $\ell^2(\mathbb{Z}_{N_1}\times\mathbb{Z}_{N_2})$.

Z uporabo zveze $e^{\imath\varphi} = \cos\varphi + \imath \cdot \sin\varphi$ lahko Fourierove bazne elemente zapišemo še na drug način s formulo

$$F_{m_1,m_2}(n_1,n_2) = \frac{1}{N_1N_2}\left(\cos\left[2\pi\left(\frac{m_1n_1}{N_1}+\frac{m_2n_2}{N_2}\right)\right] + \imath\sin\left[2\pi\left(\frac{m_1n_1}{N_1}+\frac{m_2n_2}{N_2}\right)\right]\right) .$$

Primer 2.6 Fourierova baza prostora $\ell^2(\mathbb{Z}_2 \times \mathbb{Z}_3)$ je množica

$$F = \{F_{0,0}, F_{0,1}, F_{0,2}, F_{1,0}, F_{1,1}, F_{1,2}\},$$

kjer so bazni elementi enaki:

$$F_{0,0} = \tfrac{1}{6}\cdot\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}, \qquad F_{1,0} = \tfrac{1}{6}\cdot\begin{bmatrix}1 & 1 & 1\\ -1 & -1 & -1\end{bmatrix},$$

$$F_{0,1} = \tfrac{1}{6}\cdot\begin{bmatrix}1 & -\frac{1}{2}+\frac{\imath\sqrt{3}}{2} & -\frac{1}{2}-\frac{\imath\sqrt{3}}{2}\\ 1 & -\frac{1}{2}+\frac{\imath\sqrt{3}}{2} & -\frac{1}{2}-\frac{\imath\sqrt{3}}{2}\end{bmatrix}, \quad F_{1,1} = \tfrac{1}{6}\cdot\begin{bmatrix}1 & -\frac{1}{2}+\frac{\imath\sqrt{3}}{2} & -\frac{1}{2}-\frac{\imath\sqrt{3}}{2}\\ -1 & \frac{1}{2}-\frac{\imath\sqrt{3}}{2} & \frac{1}{2}+\frac{\imath\sqrt{3}}{2}\end{bmatrix},$$

$$F_{0,2} = \tfrac{1}{6}\cdot\begin{bmatrix}1 & -\frac{1}{2}-\frac{\imath\sqrt{3}}{2} & -\frac{1}{2}+\frac{\imath\sqrt{3}}{2}\\ 1 & -\frac{1}{2}-\frac{\imath\sqrt{3}}{2} & -\frac{1}{2}+\frac{\imath\sqrt{3}}{2}\end{bmatrix}, \quad F_{1,2} = \tfrac{1}{6}\cdot\begin{bmatrix}1 & -\frac{1}{2}-\frac{\imath\sqrt{3}}{2} & -\frac{1}{2}+\frac{\imath\sqrt{3}}{2}\\ -1 & \frac{1}{2}+\frac{\imath\sqrt{3}}{2} & \frac{1}{2}-\frac{\imath\sqrt{3}}{2}\end{bmatrix}.$$

◊

2.1.2 Dvodimenzionalna diskretna Fourierova transformacija

Dvodimenzionalna diskretna Fourierova transformacija (kratica 2DFT) je preslikava

$$\mathcal{F} : \ell^2(\mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2}) \to \ell^2(\mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2}),$$

ki je za $z \in \ell^2(\mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2})$ definirana s formulo

$$\mathcal{F}(z)(m_1,m_2) \coloneqq \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} z(n_1,n_2) e^{\frac{-2\pi\imath m_1n_1}{N_1}} e^{\frac{-2\pi\imath m_2n_2}{N_2}} .$$

Dvodimenzionalna diskretna inverzna Fourierova transformacija (kratica 2IFT) je preslikava

$$\mathcal{F}^{-1} : \ell^2(\mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2}) \to \ell^2(\mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2}),$$

ki je za $w \in \ell^2(Z_{N_1} \times \mathbb{Z}_{N_2})$ definirana s formulo

$$\mathcal{F}^{-1}(w)(n_1,n_2) \coloneqq \frac{1}{N_1N_2}\sum_{m_1=0}^{N_1-1}\sum_{m_2=0}^{N_2-1} w(m_1,m_2) e^{\frac{2\pi\imath m_1n_1}{N_1}} e^{\frac{2\pi\imath m_2n_2}{N_2}} .$$

Enostavno lahko preverimo, da sta zgornji preslikavi dobro definirani in res slikata nazaj v isti prostor.

Ker je $\{E_{m_1,m_2}\}_{0\le m_1\le N_1-1, 0\le m_2\le N_2-1}$ ortonormirana baza, lahko $z \in \ell^2(\mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2})$ razvijemo po tej bazi kot

$$z = \sum_{m_1=0}^{N_1-1}\sum_{m_2=0}^{N_2-1} \langle z, E_{m_1,m_2}\rangle E_{m_1,m_2} . \tag{2.2}$$

Pri tem je

$$\langle z, E_{m_1,m_2}\rangle = \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} z(n_1,n_2)\overline{\frac{1}{\sqrt{N_1N_2}}e^{\frac{2\pi\imath m_1 n_1}{N_1}}e^{\frac{2\pi\imath m_2 n_2}{N_2}}}\ . \tag{2.3}$$

S pomočjo tega razvoja bomo preprosto pokazali, da velja naslednja trditev.

Trditev 2.7 *Za vsak $z \in \ell^2(\mathbb{Z}_{N_1}\times\mathbb{Z}_{N_2})$ velja:*

$$z = (\mathcal{F}^{-1}\mathcal{F})(z).$$

Dokaz. Z upoštevanjem formul (2.2) in (2.3) naredimo naslednji izračun:

$$\begin{aligned}
&(\mathcal{F}^{-1}\mathcal{F})(z)(k_1,k_2) = \\
&= \frac{1}{N_1N_2}\sum_{m_1=0}^{N_1-1}\sum_{m_2=0}^{N_2-1}\left(\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} z(n_1,n_2)e^{\frac{-2\pi\imath m_1 n_1}{N_1}}e^{\frac{-2\pi\imath m_2 n_2}{N_2}}\right)e^{\frac{2\pi\imath k_1 m_1}{N_1}}e^{\frac{2\pi\imath k_2 m_2}{N_2}} \\
&= \sum_{m_1=0}^{N_1-1}\sum_{m_2=0}^{N_2-1}\left(\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} z(n_1,n_2)\frac{1}{\sqrt{N_1N_2}}e^{\frac{-2\pi\imath m_1 n_1}{N_1}}e^{\frac{-2\pi\imath m_2 n_2}{N_2}}\right)\frac{1}{\sqrt{N_1N_2}}e^{\frac{2\pi\imath k_1 m_1}{N_1}}e^{\frac{2\pi\imath k_2 m_2}{N_2}} \\
&= \sum_{m_1=0}^{N_1-1}\sum_{m_2=0}^{N_2-1}\langle z, E_{m_1,m_2}\rangle E_{m_1,m_2}(k_1,k_2) = z(k_1,k_2)\ .
\end{aligned}$$

Ker ta izračun velja za vsak par $(k_1,k_2)\in\ell^2(\mathbb{Z}_{N_1}\times\mathbb{Z}_{N_2})$, je trditev s tem dokazana. ∎

Primer 2.8 Naj bo $z\in\ell^2(\mathbb{Z}_2\times\mathbb{Z}_3)$, npr.

$$z = \begin{bmatrix} 3\imath & 10 & 10-\imath \\ 2 & 0 & -\imath \end{bmatrix}.$$

Na tem primeru bomo sedaj preverili, da velja trditev 2.7. Za vsak par $(m_1,m_2)\in\mathbb{Z}_2\times\mathbb{Z}_3$ izračunamo vrednost $\mathcal{F}(z)(m_1,m_2)$ in dobimo matriko

$$\mathcal{F}(z) = w = \begin{bmatrix} 22+\imath & -8+\sqrt{3}+4\imath & -8-\sqrt{3}+4\imath \\ 18+3\imath & -12+3\imath & -12+3\imath \end{bmatrix}.$$

Če na matriki w sedaj uporabimo 2IFT, dobimo nazaj matriko z. Torej res velja $z = (\mathcal{F}^{-1}\mathcal{F})(z)$. ◊

Bolj kot razvoj po standardni bazi bo za nas zanimiv razvoj po Fourierovi bazi. Iz izračuna v zadnjem dokazu je razvidno, da je

$$z = \sum_{m_1=0}^{N_1-1}\sum_{m_2=0}^{N_2-1}\left(\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} z(n_1,n_2)\frac{1}{\sqrt{N_1N_2}}e^{\frac{-2\pi\imath m_1 n_1}{N_1}}e^{\frac{-2\pi\imath m_2 n_2}{N_2}}\right)E_{m_1,m_2}\ . \tag{2.4}$$

S preureditvijo členov in uporabo definicije Fourierove baze dobimo, da je

$$\begin{aligned}
z &= \sum_{m_1=0}^{N_1-1}\sum_{m_2=0}^{N_2-1}\left(\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} z(n_1,n_2)e^{\frac{-2\pi\imath m_1 n_1}{N_1}}e^{\frac{-2\pi\imath m_2 n_2}{N_2}}\right)F_{m_1,m_2} \\
&= \sum_{m_1=0}^{N_1-1}\sum_{m_2=0}^{N_2-1}\mathcal{F}(z)(m_1,m_2)F_{m_1,m_2}\ .
\end{aligned}$$

Kadar z razvijemo po Fourierovi bazi, pravimo, da smo ga zapisali v *frekvenčni domeni*.[1]

Primer 2.9 Element z iz primera 2.8,

$$z = \begin{bmatrix} 3\imath & 10 & 10-\imath \\ 2 & 0 & -\imath \end{bmatrix},$$

bomo zapisali v Fourierovi bazi:

$$z = (22+\imath)\cdot F_{0,0} + (-8+\sqrt{3}+4\imath)\cdot F_{0,1} + (-8-\sqrt{3}+4\imath)\cdot F_{0,2} + \\ + (18+3\imath)\cdot F_{1,0} + (-12+3\imath)\cdot F_{1,1} + (-12+3\imath)\cdot F_{1,2},$$

kjer so F_{n_1,n_2} Fourierovi bazni elementi za prostor $\ell^2(\mathbb{Z}_2 \times \mathbb{Z}_3)$. ◊

Izrek 2.10 *Za $z, w \in \ell^2(\mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2})$ veljata naslednji dve formuli:*

1. *Parsevalova formula:*
$$\langle z, w\rangle = \frac{1}{N_1 N_2}\langle \mathcal{F}(z), \mathcal{F}(w)\rangle;$$

2. *Plancherelova formula:*
$$\|z\|^2 = \frac{1}{N_1 N_2}\|\mathcal{F}(z)\|^2.$$

Dokaz. Najprej bomo dokazali Parsevalovo formulo. S primerjavo formul (2.2) in (2.4) opazimo, da velja $\mathcal{F}(z(m_1,m_2)) = \sqrt{N_1N_2}\langle z, E_{m_1,m_2}\rangle$. Z uporabo te ugotovitve in formule (2.1) za izračun skalarnega produkta dobimo, da je

$$\begin{aligned}\langle z, w\rangle &= \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} z(n_1,n_2)\overline{w(n_1,n_2)} \\ &= \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} \frac{1}{\sqrt{N_1N_2}}\mathcal{F}(z(n_1,n_2))\frac{1}{\sqrt{N_1N_2}}\mathcal{F}(w(n_1,n_2)).\end{aligned}$$

S tem smo pokazali veljavnost Parsevalove formule. Plancherelovo formulo enostavno dokažemo tako, da ponovimo postopek za $w = z$ in uporabimo definicijo norme. ∎

2.1.3 Translacijsko invariantne linearne transformacije

Transformacija je matematični zapis sistema, ki poskrbi za pretvorbo vhodnega signala v izhodnega. Sistemi, ki jih modeliramo s transformacijo, so običajno linearni (pripadajoča transformacija je linearna) in invariantni (v primeru časovno ali prostorsko zamaknjenega vhodnega signala, ima te lastnosti tudi izhodni signal). V našem delu bomo modelirali sivinske slike, zato se bomo sedaj osredotočili na dvodimenzionalne transformacije.

Naj bo sedaj z zaporedje, ki ga definiramo na množici $\mathbb{Z}\times\mathbb{Z}$ (elementi zaporedja so $z(n_1,n_2) \in \mathbb{C}$ za $(n_1,n_2) \in \mathbb{Z}\times\mathbb{Z}$). Za zaporedje z pravimo, da je *periodično* v prvi

[1] V razdelku **??** bomo videli fizikalno interpretacijo zapisa v Fourierovi bazi in naredili njihovo vizualno predstavitev.

spremenljivki s periodo N_1 in v drugi spremenljivki s periodo N_2, če za vse $n_1, n_2, j_1, j_2 \in \mathbb{Z}$ velja

$$z(n_1 + j_1 N_1, n_2 + j_2 N_2) = z(n_1, n_2) \, .$$

Definicija 2.11 Za $k_1, k_2 \in \mathbb{Z}$ je *translacijsko invariantna linearna transformacija*

$$R_{k_1,k_2} : \ell^2(\mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2}) \to \ell^2(\mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2})$$

definirana s formulo

$$(R_{k_1,k_2} z)(n_1, n_2) \coloneqq z(n_1 - k_1, n_2 - k_2) \, .$$

Pravimo, da je transformacija $T : \ell^2(\mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2}) \to \ell^2(\mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2})$ *translacijsko invariantna*, če za vse $k_1, k_2 \in \mathbb{Z}$ in $z \in \ell^2(\mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2})$ velja:

$$T(R_{k_1,k_2} z) = R_{k_1,k_2} T(z).$$

Trditev 2.12 *Če je transformacija $T : \ell^2(Z_{N_1} \times Z_{N_2}) \to \ell^2(Z_{N_1} \times Z_{N_2})$ translacijsko invariantna, tedaj za vsak Fourierov bazni element F_{m_1,m_2} velja, da je lastni vektor transformacije T.*

Dokaz. Naj bo F_{m_1,m_2} Fourierov bazni element v prostoru $\ell^2(Z_{N_1} \times Z_{N_2})$. Ker je F baza prostora $\ell^2(Z_{N_1} \times Z_{N_2})$, obstajajo taka kompleksna števila $a_{0,0}, a_{0,1} \ldots a_{0,N_2-1}, a_{1,0}$, $a_{1,1} \ldots a_{1,N_2-1} \ldots a_{N_1-1,0}, a_{N_1-1,1} \ldots a_{N_1-1,N_2-1}$, da za vsak par $(n_1, n_2) \in \mathbb{N} \times \mathbb{N}$ velja:

$$T(F_{m_1,m_2})(n_1, n_2) = \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} a_{k_1,k_2} F_{k_1,k_2}(n_1, n_2) \tag{2.5}$$

$$= \frac{1}{N_1 N_2} \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} a_{k_1,k_2} e^{\frac{2\pi\imath k_1 n_1}{N_1}} e^{\frac{2\pi\imath k_2 n_2}{N_2}} \, . \tag{2.6}$$

Po definiciji preslikave R_{k_1,k_2} velja:

$$\begin{aligned}(R_{1,0} F_{m_1,m_2})(n_1, n_2) &= F_{m_1,m_2}(n_1 - 1, n_2) = \frac{1}{N_1 N_2} e^{\frac{2\pi\imath m_1 (n_1-1)}{N_1}} e^{\frac{2\pi\imath m_2 n_2}{N_2}} \\ &= \frac{1}{N_1 N_2} e^{-\frac{2\pi\imath m_1}{N_1}} e^{\frac{2\pi\imath m_1 n_1}{N_1}} e^{\frac{2\pi\imath m_2 n_2}{N_2}} \\ &= e^{-\frac{2\pi\imath m_1}{N_1}} F_{m_1,m_2}(n_1, n_2) \, .\end{aligned}$$

Ker je T linearna preslikava in sta člena $e^{-\frac{2\pi\imath m_1}{N_1}}$ in $e^{-\frac{2\pi\imath m_2}{N_2}}$ neodvisna od spremenljivk n_1 in n_2, po enačbi (2.5), velja:

$$\begin{aligned}T(R_{1,0} F_{m_1,m_2})(n_1, n_2) &= e^{-\frac{2\pi\imath m_1}{N_1}} T(F_{m_1,m_2})(n_1, n_2) \\ &= e^{-\frac{2\pi\imath m_1}{N_1}} \frac{1}{N_1 N_2} \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} a_{k_1,k_2} F_{k_1,k_2}(n_1, n_2) \\ &= \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} e^{-\frac{2\pi\imath m_1}{N_1}} a_{k_1,k_2} F_{k_1,k_2}(n_1, n_2) \, .\end{aligned}$$

Dobili smo razvoj $T(R_{1,0}F_{m_1,m_2})(n_1,n_2)$ po Fourierovi bazi. Sedaj bomo s pomočjo (2.5) razvili po Fourierovi bazi še $(R_{1,0}T(F_{m_1,m_2}))(n_1,n_2)$:

$$\begin{aligned} R_{1,0}(T(F_{m_1,m_2}))(n_1,n_2) &= T(F_{m_1,m_2})(n_1-1,n_2) \\ &= \frac{1}{N_1N_2}\sum_{k_1=0}^{N_1-1}\sum_{k_2=0}^{N_2-1} a_{k_1,k_2} e^{\frac{2\pi\imath k_1(n_1-1)}{N_1}} e^{\frac{2\pi\imath k_2 n_2}{N_2}} \\ &= \sum_{k_1=0}^{N_1-1}\sum_{k_2=0}^{N_2-1} a_{k_1,k_2} e^{-\frac{2\pi\imath k_1}{N_1}} F_{k_1,k_2}(n_1,n_2)\ . \end{aligned}$$

Ker smo predpostavili, da je T translacijsko invariantna transformacija, za vsak par $(n_1,n_2)\in\mathbb{Z}\times\mathbb{Z}$ velja, da je $T(R_{1,0}(F_{m_1,m_2}))(n_1,n_2) = R_{1,0}(T(F_{m_1,m_2}))(n_1,n_2)$. Zaradi enoličnosti razvoja po bazi sledi, da morajo biti koeficienti v zgornjih dveh razvojih enaki. Za vsak par $(k_1,k_2)\in\mathbb{Z}_{N_1}\times\mathbb{Z}_{N_2}$ velja:

$$a_{k_1,k_2}e^{-\frac{2\pi i m_1}{N_1}} = a_{k_1,k_2}e^{-\frac{2\pi\imath k_1}{N_1}}\ . \tag{2.7}$$

Za $k_1\neq m_1$, velja $e^{-\frac{2\pi\imath m_1}{N_1}} \neq e^{-\frac{2\pi\imath k_1}{N_1}}$, ker je $0\le k_1,m_1\le N_1-1$. Če želimo, da velja enakost v zgornji enačbi, mora biti za $k_1\neq m_1$ koeficient $a_{k_1,k_2}=0$.

Podobno bi dobili, da mora biti za $k_2\neq m_2$ koeficient $a_{k_1,k_2}=0$, če bi v zgornjih izračunih namesto $R_{1,0}$ vzeli $R_{0,1}$. Dobili smo, da mora biti za $(k_1,k_2)\neq(m_1,m_2)$ koeficient $a_{k_1,k_2}=0$. V enačbi (2.5) zato odpadejo vsi členi, razen tistega pri $(k_1,k_2)=(m_1,m_2)$. Dobimo, da je $T(F_{m_1,m_2})(n_1,n_2) = a_{m_1,m_2}F_{m_1,m_2}(n_1,n_2)$ oz.

$$T(F_{m_1,m_2}) = a_{m_1,m_2}F_{m_1,m_2},$$

kar dokazuje, da je F_{m_1,m_2} lastni vektor transformacije T z lastno vrednostjo a_{m_1,m_2}. S tem je dokazana celotna trditev, saj smo to dokazali za splošen F_{m_1,m_2}. ∎

Posledica 2.13 *Translacijsko invariantne linearne transformacije T so diagonalizabilne v Fourierovi bazi.*

Definicija 2.14 Za vse $z,w\in\ell^2(Z_{N_1}\times Z_{N_2})$ in $(m_1,m_2)\in\mathbb{Z}_{N_1}\times\mathbb{Z}_{N_2}$ definiramo operator $z*w\in\ell^2(Z_{N_1}\times Z_{N_2})$ s formulo

$$z*w(m_1,m_2) \coloneqq \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} z(m_1-n_1,m_2-n_2)w(n_1,n_2)\ .$$

Primer 2.15 Naj bosta $z=\left(\begin{smallmatrix}1&1\\0&\imath\end{smallmatrix}\right)$ in $w=\left(\begin{smallmatrix}\imath&0\\1&\imath\end{smallmatrix}\right)$ elementa prostora $\ell^2(\mathbb{Z}_2\times\mathbb{Z}_2)$. Z upoštevanjem periodičnosti, izračunamo

$$\begin{aligned} z*w(0,0) &= \sum_{n_1=0}^{1}\sum_{n_2=0}^{1} z(-n_1,-n_2)w(n_1,n_2) \\ &= z(0,0)w(0,0)+z(0,-1)w(0,1)+z(-1,0)w(1,0)+z(-1,-1)w(1,1) \\ &= z(0,0)w(0,0)+z(0,1)w(0,1)+z(1,0)w(1,0)+z(1,1)w(1,1) \\ &= 1\cdot\imath+1\cdot 0+0\cdot 1+\imath\cdot\imath = \imath-1\ . \end{aligned}$$

Podobno izračunamo preostale tri vrednosti:

$$z * w(0,1) = \sum_{n_1=0}^{1}\sum_{n_2=0}^{1} z(-n_1, 1-n_2)w(n_1,n_2) = 1\cdot i + 1\cdot 0 + 0\cdot 1 + i\cdot i = i-1\ ,$$
$$z * w(1,0) = \sum_{n_1=0}^{1}\sum_{n_2=0}^{1} z(-n_1, 1-n_2)w(n_1,n_2) = 1\cdot i + 1\cdot 0 + 0\cdot 1 + i\cdot i = i-1\ ,$$
$$z * w(1,1) = \sum_{n_1=0}^{1}\sum_{n_2=0}^{1} z(-n_1, 1-n_2)w(n_1,n_2) = 1\cdot i + 1\cdot 0 + 0\cdot 1 + i\cdot i = i-1\ .$$

Torej je $z * w = ()$. ◊

Izrek 2.16 *Veljajo naslednje trditve:*

1. *Za vse* $(m_1, m_2) \in \mathbb{Z}\times\mathbb{Z}$ *velja*
$$\mathcal{F}(z * w)(m_1,m_2) = \mathcal{F}(z)(m_1,m_2)\mathcal{F}(w)(m_1,m_2).$$
2. *Za* $b \in \ell^2(Z_{N_1}\times Z_{N_2})$ *definiramo* $T_b : \ell^2(Z_{N_1}\times Z_{N_2}) \to \ell^2(Z_{N_1}\times Z_{N_2})$ *z*
$$T_b(z) = b * z.$$
Vsaki linearni transformaciji te oblike pravimo, da je konvolucijski operator. *Velja, da je* T_b *translacijsko invariantna preslikava.*
3. *Za* $m \in \ell^2(Z_{N_1}\times Z_{N_2})$ *definiramo* $T_{(m)} : \ell^2(Z_{N_1}\times Z_{N_2}) \to \ell^2(Z_{N_1}\times Z_{N_2})$ *kot*
$$T_{(m)}(z) = \mathcal{F}^{-1}(m\mathcal{F}(z)),$$
kjer je $(m\mathcal{F}(z))(n_1,n_2) = m(n_1,n_2)\mathcal{F}(z)(n_1,n_2)$ *za vsak* (n_1,n_2). *Vsaki linearni transformaciji takega tipa pravimo, da je* Fourierov multiplikativni operator. *Velja, da je vsak konvolucijski operator* T_b *enak Fourierov multiplikativni operator* $T_{(m)}$ *pri izbiri* $m = \mathcal{F}(b)$.

Dokaz. (1) Z uporabo definicij Fourierove transformacije in konvolucije naredimo naslednji izračun:

$$\begin{aligned}
&\mathcal{F}(z*w)(m_1,m_2) = \\
&= \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} z*w(n_1,n_2) e^{\frac{-2\pi\imath m_1 n_1}{N_1}} e^{\frac{-2\pi\imath m_2 n_2}{N_2}} \\
&= \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1}\sum_{k_1=0}^{N_1-1}\sum_{k_2=0}^{N_2-1} z(n_1-k_1, n_2-k_2)w(k_1,k_2) e^{\frac{-2\pi\imath m_1 n_1}{N_1}} e^{\frac{-2\pi\imath m_2 n_2}{N_2}} \\
&= \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1}\sum_{k_1=0}^{N_1-1}\sum_{k_2=0}^{N_2-1} z(n_1-k_1, n_2-k_2)w(k_1,k_2) e^{\frac{-2\pi\imath m_1 (n_1-k_1)}{N_1}} e^{\frac{-2\pi\imath m_2 (n_2-k_2)}{N_2}} e^{\frac{-2\pi\imath m_1 k_1}{N_1}} e^{\frac{-2\pi\imath m_2 k_2}{N_2}} \\
&= \sum_{k_1=0}^{N_1-1}\sum_{k_2=0}^{N_2-1} w(k_1,k_2) e^{\frac{-2\pi\imath m_1 k_1}{N_1}} e^{\frac{-2\pi\imath m_2 k_2}{N_2}} \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} z(n_1-k_1, n_2-k_2) e^{\frac{-2\pi\imath m_1 (n_1-k_1)}{N_1}} e^{\frac{-2\pi\imath m_2 (n_2-k_2)}{N_2}}\ .
\end{aligned}$$

■

Zadnjo dvojno vsoto z uvedbo novih spremenljivkk $l_1 = n_1 - k_1$ in $l_2 = n_2 - k_2$ prevedemo v:

$$\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} z(n_1-k_1, n_2-k_2) e^{\frac{-2\pi\imath m_1(n_1-k_1)}{N_1}} e^{\frac{-2\pi\imath m_2(n_2-k_2)}{N_2}} =$$
$$= \sum_{l_1=-k_1}^{N_1-1-k_1}\sum_{l_2=-k_2}^{N_2-1-l_2} z(l_1,l_2) e^{\frac{-2\pi\imath m_1 l_1}{N_1}} e^{\frac{-2\pi\imath m_2 l_2}{N_2}}$$
$$= \sum_{l_1=0}^{N_1-1}\sum_{l_2=0}^{N_2-1} z(n_1-k_1, n_2-k_2) e^{\frac{-2\pi\imath m_1 l_1}{N_1}} e^{\frac{-2\pi\imath m_2 l_2}{N_2}} \ .$$

V zadnjem enačaju smo uporabili, da sta izraza v vsoti periodična s periodo N_1 oz. N_2. Z uvedbo novih spremenljivk dobimo:

$$\mathcal{F}(z * w)(m_1, m_2) =$$
$$= \sum_{k_1=0}^{N_1-1}\sum_{k_2=0}^{N_2-1} w(k_1,k_2) e^{\frac{-2\pi\imath m_1 k_1}{N_1}} e^{\frac{-2\pi\imath m_2 k_2}{N_2}} \sum_{l_1=0}^{N_1-1}\sum_{l_2=0}^{N_2-1} z(l_1,l_2) e^{\frac{-2\pi\imath m_1 l_1}{N_1}} e^{\frac{-2\pi\imath m_2 l_2}{N_2}}$$
$$= \mathcal{F}(z)(m_1,m_2)\mathcal{F}(w)(m_1,m_2) \ ,$$

kar dokazuje prvo točko izreka, saj zgornje velja za poljubna m_1 in m_2.

(2) Naj bo $z \in \ell^2(Z_{N_1} \times Z_{N_2})$ in $(k_1, k_2) \in \mathbb{Z}\times\mathbb{Z}$. Tedaj za vsak par $(k_1, k_2) \in \mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2}$ velja naslednji izračun:

$$T_b(R_{k_1,k_2}z)(m_1,m_2) = b * (R_{k_1,k_2}z)(m_1,m_2)$$
$$= \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} b(m_1-n_1, m_2-n_2)(R_{k_1,k_2}z)(n_1,n_2)$$
$$= \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} b(m_1-n_1, m_2-n_2) z(n_1-k_1, n_2-k_2) \ .$$

Z uvedbo novih spremenljivk $l_1 = n_1 - k_1$ in $l_2 = n_2 - k_2$ v zadnji vsoti in upoštevanjem periodičnosti, za vsak par $(m_1, m_2) \in \mathbb{Z} \times \mathbb{Z}$ velja

$$T_b(R_{k_1,k_2}z)(m_1,m_2) =$$
$$= \sum_{l_1=-k_1}^{N_1-1-k_1}\sum_{l_2=-k_2}^{N_2-1-k_2} b(m_1-k_1-l_1, m_2-k_2-l_2) z(l_1,l_2)$$
$$= \sum_{l_1=0}^{N_1-1}\sum_{l_2=0}^{N_2-1} b(m_1-k_1-l_1, m_2-k_2-l_2) z(l_1,l_2)$$
$$= (b * z)(m_1-k_1, m_2-k_2)$$
$$= R_{k_1,k_2}(b * z)(m_1,m_2)$$
$$= R_{k_1,k_2}T_b(z)(m_1,m_2) \ .$$

S tem smo pokazali, da je $T_b(R_{k_1,k_2}z) = R_{k_1,k_2}T_b(z)$, tj. T_b je res translacijski operator.

(3) Z uporabo prve točke tega izreka in definicije inverzne Fourierove transformacije za vsak $z \in \ell^2(Z_{N_1} \times Z_{N_2})$ izračunamo:

$$T_b(z) = b * z = \mathcal{F}^{-1}(b * z) = \mathcal{F}^{-1}(\mathcal{F}(b)\mathcal{F}(z)) = \mathcal{F}^{-1}(m\mathcal{F}(z)) = T_{(m)}(z),$$

kar dokazuje tretjo točko izreka. ∎

Izrek 2.17 *Naj bo $T : \ell^2(Z_{N_1} \times Z_{N_2}) \to \ell^2(Z_{N_1} \times Z_{N_2})$ linearna transformacija. Naslednje trditve so ekvivalentne:*

1. *T je translacijsko invariantni operator;*
2. *T je Fourierov multiplikativni operator;*
3. *T je konvolucijski operator.*

2.1.4 Hitra Fourierova transformacija

V tem podrazdelku bomo razvili algoritem za dvodimenzionalno hitro diskretno Fourierovo transformacijo. Naj bo $z \in \ell^2(\mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2})$. Za izračun 2DFT $\mathcal{F}(z)(m_1, m_2)$ potrebujemo N_1N_2 kompleksnih množenj[2]. Za celotni izračun $\mathcal{F}(z)$ torej potrebujemo $N_1^2N_2^2$ kompleksnih množenj.

Opazimo, da se nekateri izračuni pri 2DFT ponovijo. Naslednja trditev bo izkoristila to dejstvo, s tem bomo zmanjšali število kompleksnih množenj.

Trditev 2.18 *Naj bo*

$$y(n_1, m_2) \coloneqq \sum_{n_2=0}^{N_2-1} z(n_1, n_2) e^{-\frac{2\pi \imath m_2 n_2}{N_2}}.$$

Tedaj lahko Fourierovo transformacijo zapišemo kot

$$\mathcal{F}(z)(m_1, m_2) = \sum_{n_1=0}^{N_1-1} y(n_1, m_2) e^{-\frac{2\pi \imath m_1 n_1}{N_1}}.$$

Za izračun 2DFT potrebujemo $N_1N_2(N_1 + N_2)$ kompleksnih množenj.

Dokaz. Za izračun $y(n_1, m_2)$ potrebujemo N_2 kompleksnih množenj. Skupno, za vse možne pare (n_1, m_2), torej potrebujemo $(N_1N_2) \cdot N_2$ kompleksnih množenj. Pri izračunu $\mathcal{F}(z)(m_1, m_2)$ upoštevamo, da je vrednost $y(n_1, m_2)$ že znana. Torej za njen izračun potrebujemo N_1 komplesknih množenj. Za vse možne kombinacije (m_1, m_2) skupaj potrebujemo $(N_1N_2) \cdot N_1$ kompleksnih množenj.

Celotni izračun 2DFT torej zahteva $N_1N_2(N_1 + N_2)$ kompleksnih množenj. ∎

[2]Šteli bomo le kompleksna množenja. Operacij seštevanja ne bomo šteli. Število realnih množenj je potem trikrat večje, saj za množenje dveh kompleksnih števil potrebujemo tri realna množenja: $(a+b\imath)(c+d\imath) = (a-b)d + (c-d)a + [(a-b)d + (c+d)b]\imath$.

Zgornje število kompleksnih množenj lahko še zmanjšamo. V zgornji trditvi smo izračun 2DFT prevedli na zaporedni izračun dveh enodimenzionalnih DFT. Za vsakega izmed teh izračunov torej potrebujemo največ toliko kompleksnih množenj kot jih zahteva izračun 1DFT. Izkaže se, da lahko v nekaterih primerih izračun 1DFT zelo pohitrimo, o tem se bomo prepričali v naslednjem podrazdelku.

Enodimenzionalna hitra diskretna Fourierova transformacija

Enodimenzionalno hitro diskretno Fourierovo transformacijo $\mathcal{F}$ na vektorju $z \in \ell^2(\mathbb{Z}_N)$ izračunamo po formuli:

$$\mathcal{F}(z)(m) = \sum_{n=0}^{N-1} z(n) e^{-\frac{2\pi\imath mn}{N}} .$$

S pomočjo indukcije bomo pokazali, da lahko časovno zahtevnost izračuna 1DFT izboljšamo, če izkoristimo lastnosti Fourierove transformacije. Kasneje bomo pokazali kako lahko pohitrimo izračun za splošen N, zaenkrat si poglejmo to na primeru, ko je N sodo število, torej $N = 2M$ za $M \in \mathbb{N}$.

Lema 2.19 *Naj bo $M \in \mathbb{N}$, $N = 2M$ in $z \in \ell^2(\mathbb{Z}_N)$. Definirajmo $u, v \in \ell^2(\mathbb{Z}_M)$ s formulama $u(k) = z(2k)$ in $v(k) = z(2k+1)$ za $k \in \{0, 1, \ldots, M-1\}$. Za $m \in \{0, 1, \ldots, M-1\}$ je*

$$\mathcal{F}(z)(m) = \mathcal{F}(u)(m) + e^{-\frac{2\pi\imath m}{N}} \mathcal{F}(v)(m) . \tag{2.8}$$

Za $m \in \{M, M+1, \ldots, N-1\}$ definiramo $l = m - M$, $l \in \{0, 1, \ldots, M-1\}$. Velja

$$\mathcal{F}(z)(m) = \mathcal{F}(z)(l+M) = \mathcal{F}(u)(l) + e^{-\frac{2\pi\imath l}{N}} \mathcal{F}(v)(l) . \tag{2.9}$$

Dokaz. Za $m \in \{0, 1, \ldots, N-1\}$ po definiciji velja

$$\mathcal{F}(z)(m) = \sum_{n=0}^{N-1} z(n) e^{-\frac{2\pi\imath mn}{N}} .$$

To vsoto lahko razbijemo na dve vsoti, pri čemer v prvi delni vsoti vzamemo sode indekse $n = 2k$ in v drugi delni vsoti lihe indekse $n = 2k+1$ za $k \in \{0, 1, \ldots, M-1\}$:

$$\begin{aligned}
\mathcal{F}(z)(m) &= \sum_{k=0}^{M-1} z(2k) e^{-\frac{2\pi\imath 2km}{N}} + \sum_{k=0}^{M-1} z(2k+1) e^{-\frac{2\pi\imath (2k+1)m}{N}} \\
&= \sum_{k=0}^{M-1} u(k) e^{-\frac{2\pi\imath km}{N/2}} + e^{-\frac{2\pi\imath m}{N}} \sum_{k=0}^{M-1} v(k) e^{-\frac{2\pi\imath km}{N/2}} \\
&= \sum_{k=0}^{M-1} u(k) e^{-\frac{2\pi\imath km}{M}} + e^{-\frac{2\pi\imath m}{N}} \sum_{k=0}^{M-1} v(k) e^{-\frac{2\pi\imath km}{M}} .
\end{aligned}$$

Za $m \in \{0, 1, \ldots, M-1\}$ je zadnji izraz enak $\mathcal{F}(u)(m) + e^{-\frac{2\pi\imath m}{N}} \mathcal{F}(v)(m)$, torej dobimo enakost (2.8). Za $m \in \{M, M+1, \ldots, N-1\}$ pišemo $m = l + M$ in vstavimo to v zadnji

izraz. Z upoštevanjem dejstva, da je $e^{-\frac{2\pi\imath kl}{M}}$ periodična funkcija s perido M, dobimo

$$\begin{aligned}\mathcal{F}(z)(m) &= \sum_{k=0}^{M-1} u(k)e^{-\frac{2\pi\imath k(l+M)}{M}} + e^{-\frac{2\pi\imath(l+M)}{N}} \sum_{k=0}^{M-1} v(k)e^{-\frac{2\pi\imath k(l+M)}{M}} \\ &= \sum_{k=0}^{M-1} u(k)e^{-\frac{2\pi\imath kl}{M}} + e^{-\frac{2\pi\imath l}{N}} \sum_{k=0}^{M-1} v(k)e^{-\frac{2\pi\imath kl}{M}} \ .\end{aligned}$$

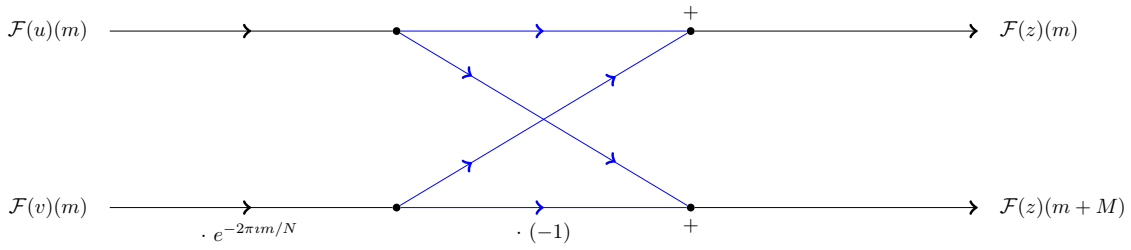
S tem smo pokazali še veljavnost enakosti (2.9). ∎

Za izračun (2.8) in (2.9) potrebujemo iste vrednosti, kar pomeni da lahko te vrednosti predhodno izračunamo in jih nato uporabimo v omenjenih izračunih. Najprej moramo torej izračunati vrednosti $\mathcal{F}(u)(l)$ in $\mathcal{F}(v)(l)$ za $l \in \{0, 1, \ldots, M-1\}$. Za vsakega izmed izračunov potrebujemo M^2 kompleksnih množenj, saj sta u in v vektorja dolžine $M = \frac{N}{2}$. Za izračun $e^{-\frac{2\pi\imath l}{N}}\mathcal{F}(v)(l)$ potrebujemo nato še dodatnih M kompleksnih množenj. Skupno število za izračun (2.8) in (2.9) je zato enako

$$M^2 + (M^2 + M) = 2M^2 + M = 2\left(\frac{N}{2}\right)^2 + \frac{N}{2} = \frac{N^2 + N}{2} \ .$$

Če uporabimo postopek iz zgornje leme, za izračun DFT potrebujemo polovico manj kompleksnih kot bi jih v primeru, ko bi računali DFT direktno po formuli. Postopek iz leme je najbolj osnovni primer hitre Fourierove transformacije.

Zanimiv je grafični prikaz postopka izračuna iz leme, saj ima diagram (glej sliko **??**), ki prikazuje postopek, obliko *metulja*. Ta izračun metulja je tako osnoven, da včasih pri rcomputer hardware štejemo koliko metuljev je bilo potebno izračunati na sekundo.



Kar smo pri postopku naredili je bilo to, da smo originalni problem izračuna Fourierove transformacije na vektorju dolžine N z nekaj računske manipulacije prevedli na podproblema velikosti $N/2$. v primeru, ko bi bila vektorja iz podproblema ponovno sode velikosti, bi postopek lahko ponovili. Najbolj zaželjen primer za izračun DFT je zato v primeru, ko je N potenca števila 2, tj. $N = 2^n$ za $n \in \mathbb{N}$. Za lažji zapis bomo uvedli oznako $E(n)$, s katero bomo označili zgornjo mejo potrebnmih kompleksnih množenj za izračun DFT vektorja dolžine N.

Lema 2.20 *Naj bo $N = 2^n$ za $n \in \mathbb{N}$. Tedaj velja*

$$E(N) \leq \frac{1}{2} N \log_2 N \ .$$

Dokaz. Dokaz naredimo z indukcijo na naravno število n. Baza indukcije je v primeru $n = 1$, ko je vektor z dolžine 2, $z = (a, b)$. Z uporabo (**??**) izračunamo $\mathcal{F}(z) = (a+b, a-b)$, za kar ne potrebujemo kompleksnih množenj. Dobimo, da je $E(2) \leq \frac{1}{2} 2 \log_2 2 = 1$, kar pomeni, da v primeru $n = 1$ lema velja. Predpostavimo sedaj, da lema velja za $n-1$ in dokažimo, da velja za n. Iz postopka v prejšnji lemi je razvidno, da je za izračun vektorja z dolžine $N = 2M$ potrebnih največ $2E(M) + M$ kompleksnih množenj. Z uporabo te opazke in indukcijske predpostavke sledi izračun

$$E(2^n) \leq 2E(2^{n-1}) + 2^{n-1} \leq 2\frac{1}{2} 2^{n-1} \log_2 2^{k-1} + 2^{k-1} = k2^{k-1} = \frac{1}{2} k2^k = \frac{1}{2} N \log_2 N \,,$$

ki pokaže, da lema velja za vsak $n \in \mathbb{N}$. ∎

Sedaj si bomo pogledali kako lahko hitro Fourierovo transformacijo uporabimo v primeru, ko N ni sodo število. V primeru, ko je N sestavljeno število, $N = pq$, lahko naredimo posplošitev zgornje leme.

Lema 2.21 *Naj bosta $p, q \in \mathbb{N}$, $N = pq$ in $z \in \ell^2(\mathbb{Z}_N)$. Za $k \in \{0, 1, \ldots, q-1\}$ definiramo zaporedje vektorjev $w_0, w_1, \ldots, w_{p-1} \in \ell^2(\mathbb{Z}_q)$ s formulo*

$$w_l(k) = z(kp + l).$$

Zaporedje vektorjev $v_0, v_1, \ldots, v_{q-1} \in \ell^2(\mathbb{Z}_p)$ definiramo za $\ell \in \{0, 1, \ldots, p-1\}$ s formulo

$$v_b(l) = e^{-\frac{2\pi \imath bl}{N}} \mathcal{F}(w)(b).$$

Tedaj za $a \in \{0, 1, \ldots, p-1\}$ in $b \in \{0, 1, \ldots, q-1\}$ velja

$$\mathcal{F}(z)(aq + b) = \mathcal{F}(v_b)(a). \tag{2.10}$$

Dokaz. Osnovni izrek o deljenju naravnih števil nam pove, da lahko vsako število $m \in \{0, 1, \ldots, N-1\}$ zapišemo v obliki $m = aq + b$ za $a \in \{0, 1, \ldots, p-1\}$ in $b \in \{0, 1, \ldots, q-1\}$. Torej znamo po lemi s formulo (2.10) izračunati $\mathcal{F}(z)(m)$ za vsak $m \in \{0, 1, \ldots, q-1\}$.

Po osnovnem izreku o deljenju lahko zapišemo vsak $n \in \{0, 1, \ldots, N-1\}$ kot $n = kp+l$ za $k \in \{0, 1, \ldots, p-1\}$ in $l \in \{0, 1, \ldots, q-1\}$. Z uporabo tega zapisa števila n in novo definiranimi zaporedji iz predpostavk leme izračunamo:

$$\begin{aligned}
\mathcal{F}(z)(aq + b) &= \sum_{n=0}^{N-1} z(n) e^{-\frac{2\pi \imath (aq+b)n}{N}} = \sum_{l=0}^{p-1} \sum_{k=0}^{q-1} z(kp + l) e^{-\frac{2\pi \imath (aq+b)(kp+l)}{pq}} \\
&= \sum_{l=0}^{p-1} e^{-2\pi \imath al} p e^{-\frac{2\pi \imath bl}{N}} \sum_{k=0}^{q-1} w_l(k) e^{-\frac{2\pi \imath bk}{q}} = \sum_{l=0}^{p-1} e^{-\frac{2\pi \imath al}{p}} e^{-\frac{2\pi \imath bl}{N}} \mathcal{F}(w_l)(b) \\
&= \sum_{l=0}^{p-1} e^{-\frac{2\pi \imath al}{p}} v_b(l) = \mathcal{F}(v_b)(a).
\end{aligned}$$

∎

Sedaj pa si poglejmo kolikšno število kompleksnih množenj je potrebno v tem primeru. Po postopku, ki je nakazan v lemi, moramo najprej izračunati vektorje $\mathcal{F}(w_l)$ za $l \in \{0, 1, \ldots, p-1\}$. Za vsak posamezen $\mathcal{F}(w_l)$ potrebujemo $E(q)$ kompleksnih množenj, skupno torej $pE(q)$. Dobljene vektorje $\mathcal{F}(w_l)(b)$ dolžine q moramo nato pomnožiti z $e^{-\frac{2\pi\imath bl}{N}}$, kar zahteva dodatnih pq kompleksnih množenj. Ko tako dobimo vektorje $v_b(a)$, moramo izračunati $\mathcal{F}(v_b)(a)$ za $b \in \{0, 1, \ldots, q-1\}$, kar zahteva dodatnih $qE(p)$ kompleksnih množenj. Skupno je torej za izračun FFT vektorja dolžine pq potrebnih

$$E(pq) \leq pE(q) + qE(p) + pq \tag{2.11}$$

kompleksnih množenj.

Kaj pa se zgodi v primeru, ko je N praštevilo. Tedaj na vektorju ne moremo uporabiti FFT. vendar pa še ni vse izgubljeno, vektor lahko namreč na koncu podaljšamo z ničlami do željene velikosti, ki omogoča izvedbo FFT. Zaradi dodanih ničel namreč ne povzročimo velike škode, izračun pa postane opazno hitrejši.

Od tu dalje bomo sedaj predpostavili, da je N potenca števila 2, tj. $N = 2^n$. Tedaj lahko $m \in \{0, 1 \ldots N-1\}$ zapišemo v obliki polinoma $m = m_0 + m_1 2 + m_2 2^2 + \ldots + m_{n-1}2^{n-1}$, kjer so $m_0, m_1, \ldots, m_{n-1} \in \{0, 1\}$. Za $z \in \ell^2(\mathbb{Z}_N)$ bomo pisali $z(m) = z(m_{n-1}, m_n, \ldots, m_1, m_0)$. Za $k = k_0 + k_1 2 + k_2 2^2 + \ldots + k_{n-1}2^{n-1}$, $k_0, k_1, \ldots, k_{n-1} \in \{0, 1\}$ izračunamo

$$\begin{aligned}
\mathcal{F}(z)(k) &= \sum_{m=0}^{N-1} z(m) e^{-\frac{2\pi\imath km}{N}} \\
&= \sum_{m_0=0}^{1} \sum_{m_1=0}^{1} \cdots \sum_{m_{n-1}=0}^{1} z(m_{n-1}, m_{n-2}, \ldots, m_1, m_0) \cdot \\
&\cdot exp\left(\frac{-2\pi\imath(k_0 + k_1 2 + k_2 2^2 + \ldots + k_{n-1}2^{n-1})(m_0 + m_1 2 + m_2 2^2 + \ldots + m_{n-1}2^{n-1})}{2^n} \right).
\end{aligned}$$

Eksponent razbijemo na produkte glede na vsoto $m_0 + m_1 2 + m_2 2^2 + \ldots + m_{n-1}2^{n-1}$, poenostavimo dobljeni izraz in ga vstavimo v formulo za $\mathcal{F}(z)(k)$. Dobimo:

$$\begin{aligned}
\mathcal{F}(z)(k) = \sum_{m_0=0}^{1} \sum_{m_1=0}^{1} \cdots \sum_{m_{n-1}=0}^{1} & z(m_{n-1}, m_{n-2}, \ldots, m_1, m_0) \cdot exp\left(\frac{-2\pi\imath k_0 2^{n-1} m_{n-1}}{2^n} \right) \\
& \cdot exp\left(\frac{-2\pi\imath(k_0 + k_1 2) 2^{n-2} m_{n-2}}{2^n} \right) \cdots exp\left(\frac{-2\pi\imath(k_0 + k_1 2 + \cdots + k_{n-1}2^{n-1}) m_0}{2^n} \right).
\end{aligned}$$

Notranja vsota je odvisna od spremenljivk $m_0, m_1, \ldots, m_{n-2}$ in k_0, nič pa od $k_1, k_2, \ldots, k_{n-1}$. Lahko definiramo

$$\begin{aligned}
& y_1(k_0, m_{n-2}, m_{n-3}, \ldots, m_0) \\
&= \sum_{m_{n-1}=0}^{1} z(m_{n-1}, m_{n-2}, \ldots, m_1, m_0) \cdot \exp\left(-\frac{2\pi\imath k_0 2^{n-1} m_{n-1}}{2^n} \right) \\
&= z(0, m_{n-2}, \ldots, m_1, m_0) \cdot 1 + z(1, m_{n-2}, \ldots, m_1, m_0) \cdot \exp\left(-\frac{2\pi\imath k_0 2^{n-1}}{2^n} \right) .
\end{aligned}$$

Izračun y_1 zahteva le eno kompleksno množenje za vsako izmed 2^n možnih vrednosti parametrov $k_0, m_{n-2}, m_{n-1}, \ldots, m_1, m_0$. Za izračun vseh možnih vrednosti izraza y_1 potrebujemo torej 2^n kompleksnih množenj.

Na naslednjem koraku lahko sedaj izračunamo y_2 s formulo

$$\begin{aligned} &y_2(k_0, k_1, m_{n-3}, \ldots, m_0) \\ &= \sum_{m_{n-1}=0}^{1} y_1(k_0, m_{n-2}, \ldots, m_1, m_0) \cdot \exp\left(-\frac{2\pi\imath(k_0 + k_1 2)2^{n-2} m_{n-2}}{2^n}\right) . \end{aligned}$$

Ta izračun prav tako potrebuje 2^n kompleksnih množenj. Naprej nadaljujemo podobno. Poglejmo si shemo, ki prikazuje kako izračunamo z:

$$\begin{aligned} z(m_{n-1}, m_{n-2}, \ldots, m_1, m_0) &\to y_1(k_0, m_{n-2}, \ldots, m_1, m_0) \\ y_1(k_0, m_{n-2}, m_{n-3}, \ldots, m_0) &\to y_2(k_0, k_1, m_{n-3}, \ldots, m_0) \\ &\vdots \\ y_{n-1}(k_0, k_1, k_2, \ldots, k_{n-2}, m_0) &\to y_n(k_0, k_1, k_2, \ldots, k_{n-2}, k_{n-1}) . \end{aligned}$$

Kot smo rekli, na vsakem koraku potrebujemo 2^n kompleksnih množenj, imamo n korakov, skupno torej $n \cdot 2^n = N \log_2 N$ kompleksnih množenj. Poleg tega opazimo tudi, da po izračunu y_j vektorja y_{j-1} ne potrebujemo več. IZračune vektorje lahko torej delamo na mestu, kar pomeni, da lahko na vsakem koraku prejšnje podatke zamenjamo z novimi trenutnimi podatki.

Prav tako lahko z $\frac{N}{2} \log_2 N$ kompleksnimi množenji izračunamo IDFT po formuli $\mathcal{F}^{-1}(w)(n) = \frac{1}{N}\mathcal{F}(w)(N-n)$.

S pomočjo DFT in IDFT lahko sedaj pospešimo tudi računanje konvolucije. Vemo namreč, da velja:

$$z * w = \mathcal{F}^{-1}(\mathcal{F}(z)\mathcal{F}(w)).$$

Skupno porabimo za izračun konvolucije $N + \frac{3N}{2} \log_2 N$ kompleksnih množenj.

Dvodimenzionalna hitra diskretna Fourierova transformacija

Direkten izračun vseh vrednosti $y(n_1, m_2)$ zahteva $N_1 N_2^2$ kompleksnih množenj, za tem pa direkten izračun vseh vrednosti za $\hat{z}(m_1, m_2)$ zahteva $N_1^2 N_2$ kompleksnih množenj. Skupaj torej potrebujemo $N_1 N_2 (N_1 + N_2)$ množenj.

Opomba 2.22 Razlog za to izboljšavo računske zahtevnosti je ta, da 2D DFT sestoji, po definiciji, iz izračuna ene transformacije veliksoti N_2 (notranja vsota), ki ji sledi izračun transformacije velikosti N_1. Izračunu, ki ga lahko razbijemo na zaporedje manjših vsot na ta način, pravimo *paraleliziran*. To, da je FFT paraleliziran izračun sledi iz dejstva, da je DFT paraleliziran izračun.

Trditev 2.23 *Naj bosta N_1 in N_2 števili potence 2. Z uporabo računanja od zgoraj namesto direktnega računanja, se da $\mathcal{F}(z)$ izračunati z uporabo največ $\frac{1}{2} N_1 N_2 \log_2(N_1 N_2)$ kompleksnih množenj.*

2.2 Transformacije slik

2.2.1 Signali

Signali so funkcije ene ali več spremenljivk, ki nam posredujejo določene informacije o nekem fizikalnem pojavu, npr. podatke o zvočni frekvenci glasbila. Podatke, ki bi jih želeli dobiti iz vhodnega signala, največkrat ne dobimo direktno, temveč moramo vhodni signal pred njegovo analizo pretvoriti v nek drug sistem oz. zapis. Matematično povedano, na funkciji (matematični predstavitvi signala) moramo izvesti neko transformacijo. Predstavitev signala s funkcijo in transformacije, ki jih izvedemo na tej funkciji, so odvisne od lastnosti, ki jim preučevani signal ustreza (periodičnost, končnost, zveznost ...). Slika je signal, ki ga uvrščamo v skupini diskretnih in digitalnih signalov. Če imamo sivinsko sliko, potem je osnovna predstavitev signala $n \times m$ matrika, ki ima vrednosti elementov z intervala $[0, 1]$. Pri tem sta n višina in m širina slike, vrednosti elementov pa nastavimo na vrednosti pripadajočih slikovnih točk. Sliko bomo označili z I, njene slikovne točke pa z $I(x, y)$.

Transformacijo slike I bomo označili s T in jo definirali kot:

$$T : I(x, y) \to J(u, v).$$

Obravnavali bomo tri skupine transformacij:

1. točkovne transformacije (izhodna vrednost transformacije je odvisna le od ene vrednosti; ni nujno, da uporabimo pri izračunu vrednosti;

2. lokalne transformacije (izhodna vrednost transformacije T je odvisna od vseh vrednosti v soseski pripadajoče slikovne točke vhodne slike);

3. globalne transformacije (izhodna vrednost transformacije T v točki (x, y) je odvisna od vseh vrednosti vhodne slike).

V nadaljevanju bomo spoznali posamezne skupine teh transformacij in njihovo praktično vrednost.

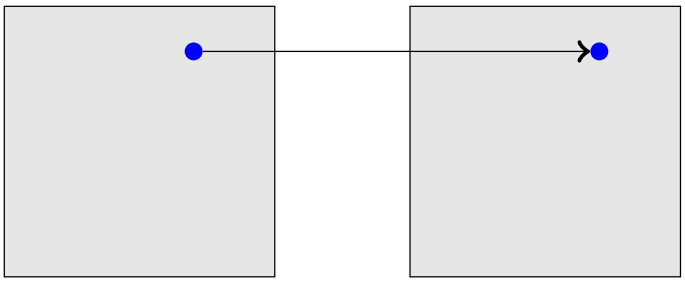
2.2.2 Točkovne transformacije

Točkovne transformacije pri izračunu vrednosti $T(x, y)$ uporabijo le podatek o vrednosti slikovne točke (x, y) (glej sliko 2.1). Za izračun točkovne transformacije T na $n \times m$ veliki sliki I porabimo $\mathcal{O}(k \cdot nm)$ časa, pri čemer je $O(k)$ časovna zahtevnost izračuna vrednosti $T(x, y)$ za eno slikovno točko. Ker za izračun potrebujemo le vrednost dotične slikovne točke, lahko izvedemo računanje na mestu, torej pri računanju ne potrebujemo dodatnega pomnilnika.

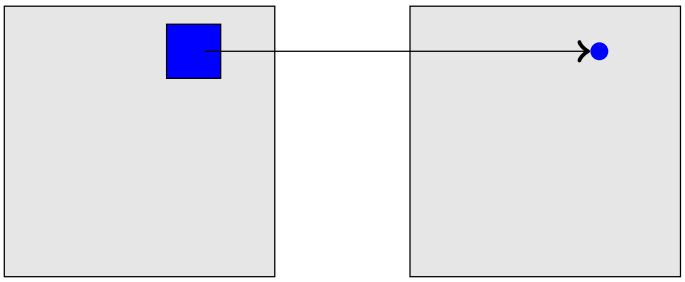
Enostavni primeri točkovnih transformacij so: rotacija slike, zrcaljenje, skrčitev, ... Za nas pomembne točkovne transformacije so tiste, ki spreminjajo barve itd.

2.2.3 Lokalne transformacije

Pri lokalnih transformacijah za izračun $T(x, y)$ v slikovni točki (x, y) uporabimo vrednosti slikovnih točk v njeni soseki (glej sliko 2.2).



Slika 2.1: Točkovna transformacija vhodne slike.



Slika 2.2: Lokalna transformacija na sliki.

Na področju obdelave slik so tovrstne transformacije uveljavljen postopek, ki mu pravimo *filtriranje*. V tem kontekstu je $T(x, y)$ utežena vsota vrednosti slikovnih točk iz pravokotne okolice slikovne točke (x, y). Uteži shranimo v *filtrirno masko* h, ki je $(2b + 1) \times (2a + 1)$ matrika z realnimi vrednostmi. Običajno so filtrirne maske kvadratne in manjših velikosti. Vrednosti $T(x, y)$, ki jih dobimo pri filtriranju slike I s filtrom h, izračunamo s pomočjo konvolucije:

$$T(x,y) = \frac{(I * h)(x,y)}{\sum_{s=-a}^{a}\sum_{t=-b}^{b} h(s,t)} = \frac{\sum_{s=-a}^{a}\sum_{t=-b}^{b} h(s,t)I(x-s,y-t)}{\sum_{s=-a}^{a}\sum_{t=-b}^{b} h(s,t)} \,. \tag{2.12}$$

Opomba 2.24 V literaturi velikokrat poudarijo, da gre za filtriranje v slikovni domeni, saj poznamo tudi filtriranje v Fourierovi domeni, ki pa ga bomo mi spoznali malce kasneje.

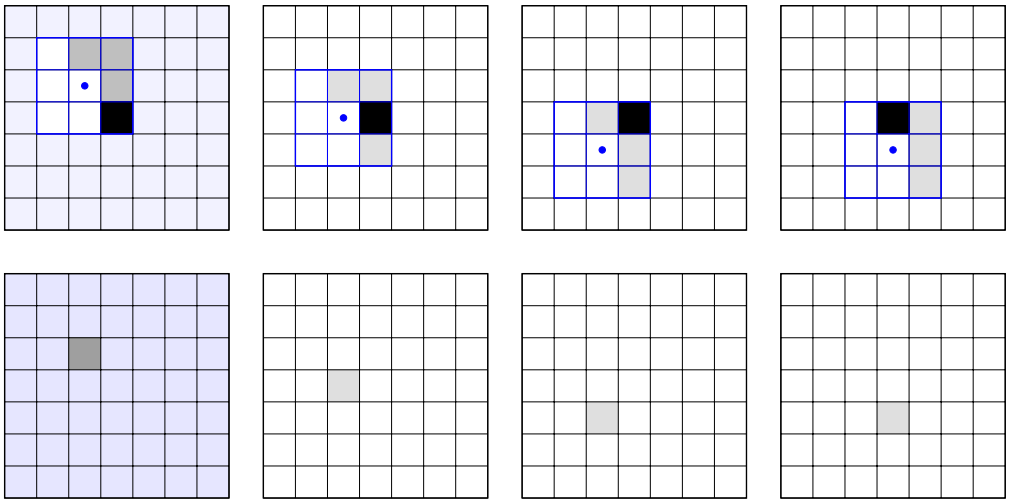
Opomba 2.25 Kot vidimo v enačbi (2.12), smo rezultat, ki ga dobimo pri konvoluciji slike in filtrirne maske, normirali. Namesto tega bi lahko za filtrirno masko zahtevali, da je normirana (tj. ima vsoto vrednosti 1). Filtrirne maske, ki jih bomo predstavili v našem delu, bomo zapisali v normirani obliki.

Postopek filtriranja obsega večji del področja pri obdelavi slik. Z uporabo filtrirnih mask lahko poiščemo, zgladimo ali izostrimo robove na sliki, zameglimo celotno sliko ... V nadaljevanju bomo predstavili primere filtrirnih mask, ki jih bomo kasneje tudi uporabljali v algoritmih za likovno upodabljanje slik.

Tehnični pogled na filtriranje

Za boljšo predstavo bomo sedaj filtriranje predstavili še grafično. Filtrirno masko h (glej **??**) najprej rotiramo za 180°, da dobimo filtrrino masko h'. Slednjo nato postavimo na vhodno sliko I tako, da center maske sovpada s slikovno točko na sliki I, v kateri želimo izračunati vrednost $T(x, y)$. Vrednost $T(x, y)$ sedaj izračunamo tako, da pomnožimo istoležne vrednosti v matriki h' in na sliki I (glej sliko **??**). Za razliko od točkovne transformacije, pri kateri smo novo vrednost lahko kar shranili na mesto (x, y) v matriki I, moramo naračunane vrednosti shraniti v novo matriko J.

V primeru, ko želimo izračunati npr. vrednost $T(x, y)$ za slikovno točko v kotu slike, del filtrirne maske ne pokriva slike I. Da se izognemu temu problemu, sliko I, pred filtriranjem s filtrirno masko velikosti $(2a + 1) \times (2a + 1)$, obrobimo naokoli z a celicami. Njihove vrednosti lahko nastavimo na več načinov (za primer glej sliko **??**). Pri tem opazimo, da lahko sedaj (ob privzetku, da filtrirane vrednosti računamo po vrsti) namesto, da za novo naračunane vrednosti ustvarimo novo matriko J, pri računanju vrednosti $T(x, y)$ novo vrednost shranimo na mesto $(x - a, y - 1)$, saj te vrednosti ne bomo več potrebovali pri izračunu ostalih vrednosti. S tem v primeru velikih slik naredimo velik prihranek pri pomnilniku.



Slika 2.3: Filtriranje v slikovni domeni.

Primer 2.26

Omenimo naj še, da so filtrirne maske, za razliko od maske v zgornjem primeru, običajno simetrične. Pri računanju filtriranih vrednosti lahko zato preskočimo rotacijo filtrirne maske, saj dobimo nazaj isto filtrirno masko.

Zgoraj predstavljeni postopek je osnovni način filtriranja. V naslednjem podrazdelku, kjer bomo izpeljali filter za zameglitev slike, pa bomo spoznali še drug pristop k filtriranju.

Zamegljenje slike

Impulzna slika $\Delta_{p,q}$ je slika, ki ima vrednosti vseh slikovnih točk, razen ene z vrednostjo 1, enake 0. Definirana je kot:

$$\Delta_{p,q}(x-p, y-q) = \begin{cases} 1 & \text{če } (x,y) = (p,q), \\ 0 & \text{sicer.} \end{cases}$$

Konvolucijo slike I z w uteženo impulzno sliko $\Delta_{p,q}$ izračunamo kot:

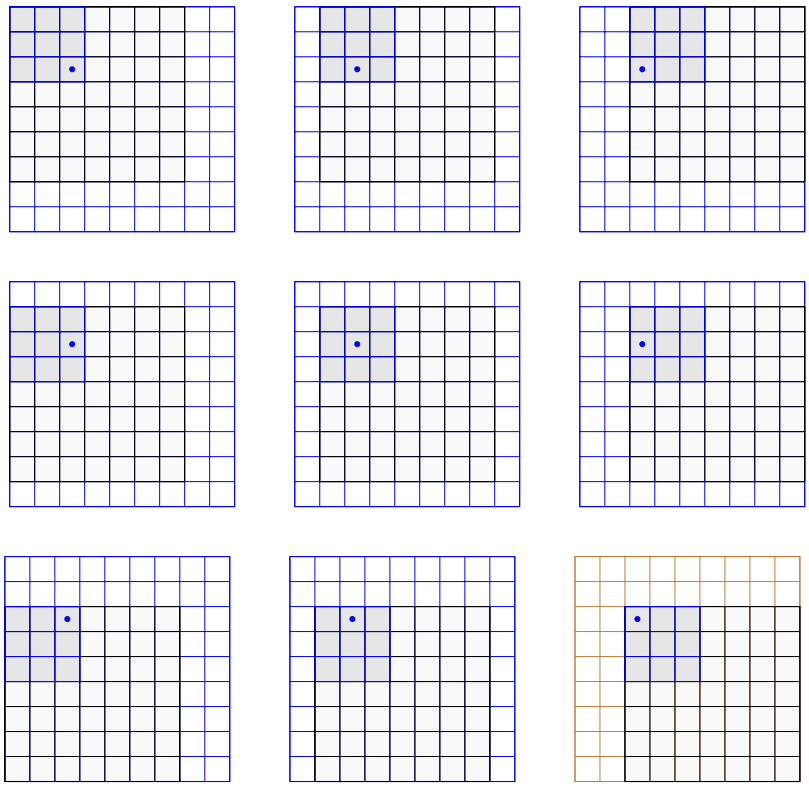
$$[I * w\Delta_{p,q}](x,y) = w \cdot I(x-p, y-q) \ .$$

Postopek za izračun filtrirane slike:

1. Za vsako celico filtrirne maske h naredimo eno prazno matriko velikosti $(h+2a) \times (w+2a)$.

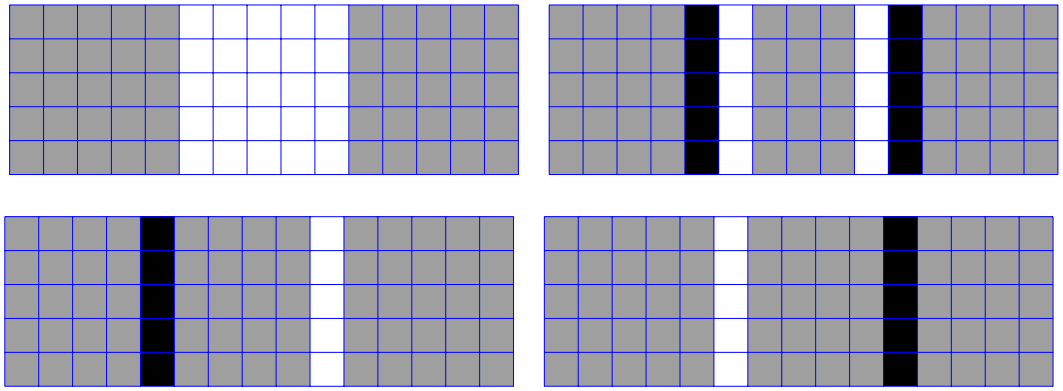
2. V vsako izmed teh praznih matrik kopiramo vhodno sliko I tako, da

Na sliki **??**

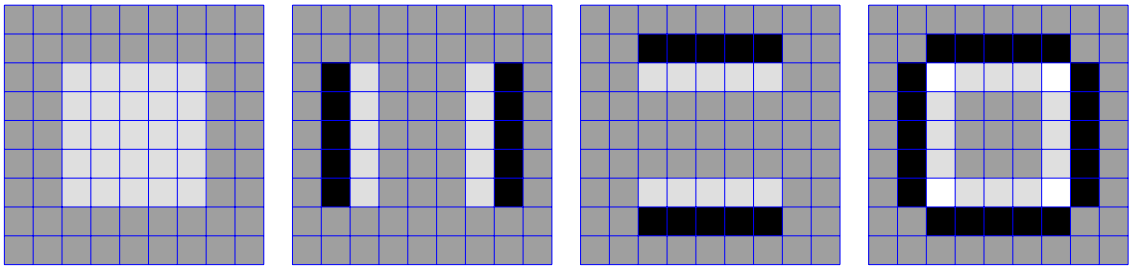


Slika 2.4: Filtriranje v slikovni domeni.

Primer 2.27



Slika 2.5: Robovi.



Slika 2.6: Robovi v kvadratu.

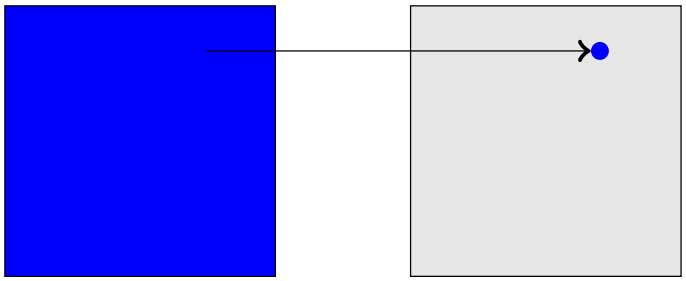
Filter za zaznavo robov na sliki

Kaj je rob ?

Ne maram ljudi.

2.2.4 Globalne transformacije

Pri globalnih transformacijah za izračun vrednosti $T(x, y)$ upoštevamo vrednosti vseh slikovnih točk na sliki (glej sliko 2.7). Najbolj pomembna transformacija te vrste je za nas Fourierova transformacija. V tem podrazdelku si bomo pogledali kakšna je praktična vrednost Fourierove transformacije pri obdelavi slik.



Slika 2.7: Globalna transformacija.

Kot smo videli v prvem razdelku tega poglavja lahko sliko I, namesto v običajni

evklidski bazi, zapišemo v Fourierovi bazi:

$$I(x,y) = \sum_{v=0}^{N_1-1} \sum_{u=0}^{N_2-1} \mathcal{F}(I)(u,v) F_{u,v}(x,y) \ .$$

Za vizualno predstavitev slike bomo na Fourierovo transformacijo pogledali z nekoliko bolj fizikalnega vidika. Za začetek si poglejmo dvodimenzionalne sinusoide.

Dvodimenzionalne sinusoide

Zaradi poenostavitve zapisa naj velja $N_1 = N_2 = N$. Tedaj bomo zapisali $F_{u,v}(x,y)$ v polarni obliki:

$$F_{u,v}(x,y) = e^{\pm \imath \frac{2\pi\omega}{N} \cdot (u \sin\theta + v\cos\theta)},$$

kjer je $y = \omega \sin\theta$, $x = \omega\cos\theta$ in $\omega = \sqrt{x^2+y^2}$. Če pišemo še $\lambda = \frac{N}{\omega}$, dobimo

$$F_{u,v}(x,y) = \cos[\frac{2\pi}{\lambda}(u\sin\theta + v\cos\theta)] \pm \imath \sin[\frac{2\pi}{\lambda}(u\sin\theta + v\cos\theta)].$$

Tako realni del $F_{u,v}(x,y)$

$$\cos[\frac{2\pi}{\lambda}(u\sin\theta + v\cos\theta)]$$

kot imaginarni del

$$\pm \sin\left[\frac{2\pi}{\lambda}(u\sin\theta + v\cos\theta)\right]$$

sta sinusoidi z amplitudo 1, periodo λ in smerjo θ. Tedaj je $\frac{2\pi\omega}{N}$ radialna frekvenca, $\frac{\omega}{N}$ je frekvenca in $\lambda = \frac{N}{\omega}$ je dolžina vala sinusoide, merjena v slikovnih točkah.

Dvodimenzionalno sinusoido bomo zapisali s formulo

$$\frac{A}{2} \cdot \left(\cos\left[\frac{2\pi}{\lambda} \cdot (u \cdot \sin\Phi + v \cdot \cos\Phi) + \phi\right] + 1\right),$$

kjer je A amplituda sinusoide in ϕ je fazni zamik.

Definicija 2.28 Naj bosta Re in Im realni in kompleksni del vrednosti $\mathcal{F}(I)(u,v)$. Definiramo pojme:

1. *Fourierov spekter*:

$$|\mathcal{F}(I)(u,v)| = \sqrt{Re^2(u,v) + Im^2(u,v)};$$

2. *fazni zamik*:

$$\Phi(u,v) = \arctan\frac{Im(u,v)}{Re(u,v)};$$

3. *močnostni spekter*:

$$P(u,v) = Re^2(u,v) + Im^2(u,v).$$

Fourierovo transformacijo lahko z novo definiranimi pojmi zapišemo v polarni obliki:

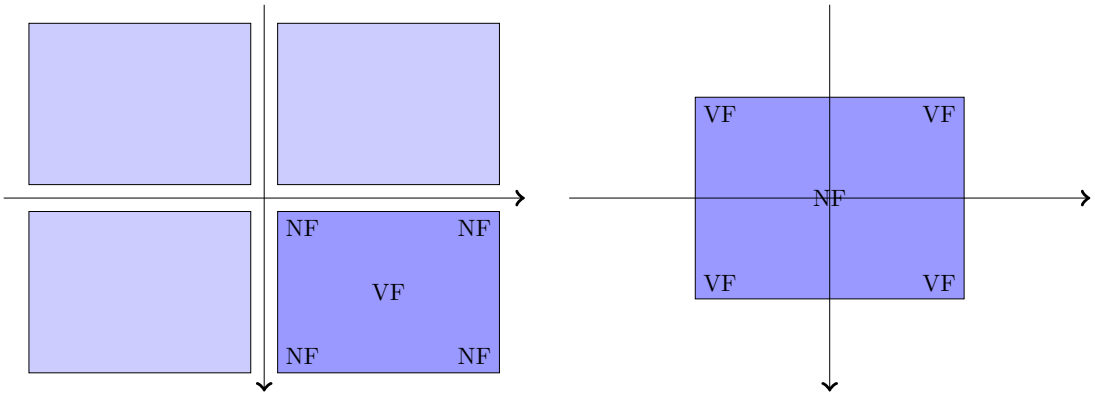
$$\mathcal{F}(z)(u,v) = |\mathcal{F}(z)(u,v)|\, e^{-\imath\cdot\Phi(u,v)}.$$

Poveš, da če je pikica v frekvenčni domeni je to sinus v navadni evklidski ravnini.

Primer 2.29 Primer slike in njenega Fouirerovega spektra. In primer ene slike, ki bo neka vsota, nekaj nekaj baznih elemntov.

Primer 2.30 Poglejmo si primer, ko imamo isti spekter in različni fazni zamik; ter isti fazni zamik, a različni spekter. Fazni zamik spreminjamo tako, da množimo z matriko R. Dobimo razne slike, ena izmed je lepa, ostale so kar nekaj ...

O spektru in faznem zamiku bomo govorili v nadaljevanju, saj sta to informaciji, ki nam povesta ogromno o sliki. Sedaj pa si najprej poglejmo še o translacijah in rotacijah Fourierove transformacije ...



Slika 2.8: Originalna DFT.

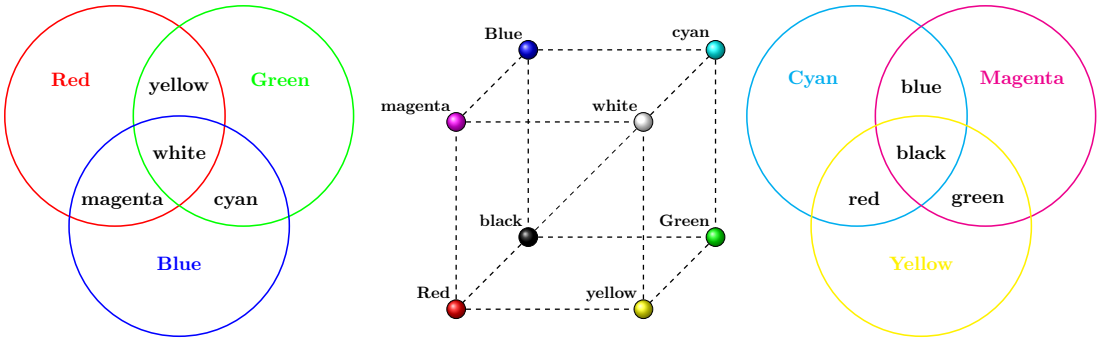
2.3 Barvni modeli

Poznamo različne barvne modele:

- RGB barvni model;
- CMY barvni model;
- HSV barvni model;
- HSL barvni model;
- YUV barvni model.

2.3.1 RGB in CMY barvna modela

RGB barvni prostor predstavimo z enotsko kocko. Oglišča kocke so enaka:



(a) (b) (c)

Slika 2.9: RGB kocka.

2.3.2 HSV in HSL barvna modela

To je pa polarna predstavitev RGB barvnega modela.

2.3.3 Kubelka–Monk barvni model

2.4 Odstranjevanje šuma na sliki

Odstranjevanje (digitalnega) šuma predstavlja pomembno področje pri obdelavi slik, saj ta običajno povzroča motnje in nepravilnosti pri delovanju algoritmov, ki jih uporabljamo za obdelavo slik. Pri tem naj opozorimo, da s pojmom šum mnogi poimenujejo tudi fine strukture in teksture, ki so prisotne na sliki. Slednje namreč tudi predstavljajo težave, vendar pa se moramo njim izogniti pri samem algoritmu za obdelavo slik.

V tem razdelku bomo predstavili merilo uspešnosti za odpravljanje digitalnega šuma na sliki, ki je predstavljen v članku [3]. Nadalje bomo primerjali različne metode, ki so predstavljene v tem članku, in nato metodo nelokalnega odstranjevanja šuma dopolnili z njeno pohitritvijo, opisano v članku [17]. V zadnjem delu bomo nato predstavili še implementacije posameznih metod in primerjali eksperimentalno dobljene podatke o učinkovitosti posameznih metod.

2.4.1 Metoda šuma

Naj bo I vhodna slika, na kateri je prisoten šum. Zaradi poenostavitve sintakse bomo slikovne točke, namesto z (x, y), označili z $i = (x_i, y_i)$. Vrednosti slikovnih točk na sliki I bomo označili z $v(i)$. Slednjo vrednost lahko zapišemo kot vsoto $v(i) = u(i) + n(i)$, pri čemer $u(i)$ predstavlja vrednost slikovne točke na sliki brez prisotnega šuma in $n(i)$ predstavlja vrednost prisotnega šuma. Z v bomo označili množico vrednosti $v = \{v(i) \mid i \in I\}$, ki jo poimenujemo *realna slika*. *Originalna slika* je množica vrednosti $u = \{u(i) \mid i \in I\}$. Množica $n = \{n(i) \mid i \in I\}$ pa je pristoni *šum* na sliki. Za modeliranje prisotnosti digitalnega šuma na sliki je najboljša izbira *Gaussov beli šum*, tj. Gaussove vrednosti s srednjo vrednostjo 0 in standardnim odklonom σ^2.

Definicija 2.31 *Operator šuma* D_h je tak operator, za katerega lahko naredimo dekom-

pozicijo realne slike v:

$$v = D_h v + n(D_h, v),$$

kjer je parameter h odvisen od standardnega odklona šuma.

Od operatorja šuma pričakujemo, da bo množica (slika) $D_h v$ gladkejša od originalne slike. Za dobljeni šum $n(D_h v)$ pa želimo, da čimbolje aproksimira beli šum. Nočemo pa, da bi poleg digitalnega šuma na sliki operator šuma odstranil tudi morebitno prisotno teksturo in pokvaril videz drobnih detajlov na sliki. Ker večina metod deluje na principu povprečenja sosednjih vrednosti slikovnih točk, pa zgornje zahteve povečini pri metodah ne držijo v celoti. Za merilo uspešnosti posameznih metod oz. operatorja šuma bomo uvedli t. i. *metodo šuma.*

Definicija 2.32 Naj bo u originalna slika in D_h operator šuma, ki je odvisen od parametra h. *Metoda šuma* je definirana kot razlika

$$u - D_h u.$$

Metoda šuma torej predstavlja razliko med originalno sliko u (na kateri ni nujno prisoten šum) in $D_h u$ (originalna slika, na kateri smo uporabili operator šuma). V primeru, ko bi originalna slika vsebovala neko teksturo, vzorce ali določeno strukturo, se te v metodi šuma ne smejo pojaviti, saj bi to kazalo na to, da smo pokvarili videz originalne slike. Metoda šuma mora izgledati kot aproksimacija šuma, celo v primerih, ko je na originalni sliki šum prisoten v zelo majhnih količinah.

2.4.2 Lokalni algoritmi za odstranjevanje šuma na sliki

Predstavili bomo tri metode za odstranjevanje šuma, ki pri računanju operatorja šuma izkoriščajo le lokalne podatke.

Gaussovo filtriranje

Gaussovo filtriranje sodi v skupino izotropnega linearnega filtriranja slik, kjer naredimo konvolucijo slike z linearnim simetričnim filtrom. Gaussov filter definiramo s funkcijo

$$G_h(x) := \frac{1}{4\pi h^2} e^{-\frac{|x|^2}{4h^2}} .$$

V tem primeru ima G_h standardni odklon h in velja naslednji izrek.

Izrek 2.33 *Metoda šuma za Gaussovo filtriranje je za dovolj majhen h enaka*

$$u - G_h * u = -h^2 \Delta u + o(h^2) .$$

Zgornji izrek dokažemo s pomočjo toplotne enačbe, saj izkoristimo dejstvo, da je fundamentalna rešitev toplotne enačbe porazdeljena po Gaussu (dokaz lahko bralec poišče v [20]).

Izrek nam pove, da bo v harmoničnih delih slike metoda šuma enaka 0, medtem ko bo v bližini linij, robov in delov tekstur imela visoke vrednosti. Na zveznih območjih slike bo metoda šuma torej optimalna, saj bo v celoti odstranila prisotni šum in pri tem ohranila videz slike. Deli slike v okolici linij, robov in prisotne teksture na sliki bodo po odstranitvi šuma postali zamegljeni in s tem pokvarili videz slike.

Anizotropno filtriranje

Ideje za to vrsto filtriranja so bile predstavljene v [27]. Pri anizotropnem filtriranju (kratica AF) poskušamo dejstvo, da pri Gaussovem filtriranju dobimo zamegljene dele slike, popraviti s tem, da v slikovni točki i naredimo konvolucijo s filtrom G_h le v smeri, ki je ortogonalna na $G(i)$. Pri tem je $G = \sqrt{(\partial_x u)^2 + (\partial_y u)^2}$, $\partial_x u$ in $\partial_y u$ sta gradientni sliki v x in y smeri. Konvolucijo naredimo torej le v "smeri linije, ki poteka skozi slikovno točko i."

Operator šuma za AF je za x, $G(x) \neq 0$, definiran s funkcijo

$$AF_h u(x) = \int G_h(t) \cdot u\left(x + t\frac{G(x)^\perp}{|G(x)|}\right) dt \, .$$

Ob predpostavki, da je originalna slika u dvakrat zvezno odvedljiva v x, lahko s pomočjo Taylorjeve vrste drugega reda pokažemo veljavnost naslednjega izreka (dokaz bomo tukaj izpustili).

Izrek 2.34 *Metoda šuma za anizotropni filter AF je na območjih, kjer velja $Du(x) \neq 0$, enaka*

$$u(x) - AF_h u(x) = -\frac{1}{2}h^2 \, |G| \cdot \kappa(u)(x) + o(h^2) \, .$$

V metodi šuma se pojavi ukrivljenost krivulje $\kappa(u)(x)$, saj je uspešnost te metode odvisna od ukrivljenosti krivulje, ki poteka skozi točko. V bližini slikovnih točk, kjer je linija, ki poteka skozi, lokalno ravna, je metoda šuma enaka 0. Posledično se ravne linije in robovi pri tej vrsti filtriranja šuma zelo lepo ohranijo. V okolici slikovnih točk, kjer imajo linije veliko ukrivljenost in so vrednosti gradienta velike, pa metoda šuma doseže velike vrednosti. Posledično na zveznih delih slike in v okolici linij, ki imajo veliko ukrivljenost, dobimo zamegljeno sliko.

Sosedsko filtriranje

Sosedsko filtriranje je skupno ime za skupino filtriranj, pri katerih novo vrednost slikovne točke i določimo na podlagi povprečja vrednosti tistih slikovnih točk iz njene soseske, ki imajo podobno vrednost.

Leta 1985 je Yaroslavsky v [38] predstavil filter te vrste. Operator šuma je za $x \in \Omega$ definiral s funkcijo

$$YNF_{h,\rho} u(x) \coloneqq \frac{1}{C(x)} \int_{B_\rho(x)} u(y) e^{-\frac{|u(y)-u(x)|^2}{h^2}} dy \, ,$$

kjer je normalizacijska konstanta definirana kot $C(x) = \int_{B_\rho(x)} e^{-\frac{|u(y)-u(x)|^2}{h^2}} dy$. Za sosesko slikovne točke x je izbral fiksno okolico $B_\rho(x)$, katere velikost je odvisna od parametra ρ. S filtrirnim parametrom h pa nadzira katere vrednosti slikovnih točk bomo vzeli v povprečje. Ker pri računanju povprečja uporabimo le slikovne točke s podobno vrednostjo, se na ta način izognemu zamegljenju linij in robov.

Deset let kasneje je bil predstavljen bolj prepoznavni filter t. i. SUSAN [32], pri katerem ne fiksiramo območja po katerem integriramo, temveč integrand pomnožimo z izrazom $e^{-\frac{|y-x|^2}{\rho^2}}$. V tem izrazu, podobno kot pri YNF, s parametrom ρ določimo soseko slikovnih

točk, ki jih bomo upoštevali pri računanju povprečja. Operator šuma je definiran s funkcijo

$$SNF_{h,\rho}u(x) \coloneqq \frac{1}{C(x)} \int_{\Omega} u(y) e^{-\frac{|y-x|^2}{\rho^2}} e^{-\frac{|u(y)-u(x)|^2}{h^2}} dy \ ,$$

kjer je normalizacijska konstanta definirana kot $C(x) = \int_{\Omega} e^{-\frac{|y-x|^2}{\rho^2}} e^{-\frac{|u(y)-u(x)|^2}{h^2}}$.

V praksi se operatorja šuma $YNF_{h,\rho}$ in $SNF_{h,\rho}$ obnašata podobno. Uporabljeni pristop za izbiro slikovnih točk, ki jih bomo upoštevali pri povprečju, v okolici linij in robov poskrbi, da pri računanju ne bomo hkrati vzeli v povprečje vrednosti, ki nastopijo na robu in v njegovi bližini. Na ta način se izognemu pojavu zamegljenosti robov po filtriranju. Vendar pa sosedski filter odpove v primeru, ko je v slikovnih točkah, ki jih primerjamo med seboj, prisoten šum. Tedaj namreč dobimo lažno območje slikovnih točk s podobno vrednostjo. Slednje vodi do umetnega pojava nepravilnosti na filtrirani sliki. Problem nastane zato, ker v integrandu primerjamo le vrednosti dveh slikovnih točk, nič pa ne upoštevamo geometrijske sestave njunih okolic. Na ta način ne zaznamo, da je v teh dveh slikovnih točkah prisoten šum, ki daje lažno prepričanje, da sta vrednosti podobni. Natančno opredeljen izrek, ki podpira trditve v tem odstavku, lahko bralec skupaj z dokazom najde v [4, strani 10–13].

2.4.3 Nelokalni algoritem za odstranjevanje šuma na sliki

Naj bo dana realna slika $v = \{v(i) \mid i \in I\}$. Za razliko od prejšnih metod, kjer smo vrednost za slikovno točko i naračunali le na podlagi vrednosti, ki se nahajajo v njeni bližini, bomo sedaj pri računanju uporabili vrednosti vseh slikovnih točk. S tem bomo sicer zelo povečali računsko zahtevnost algoritma, vendar bomo slednjo nato v praksi zmanjšali s premišljeno uporabo nekaterih podatkovnih struktur in omejitvijo območja, ki ga bomo uporabili pri računanju vrednosti v slikovni točki i.

Predstavitev postopka

Najprej bomo definirali vrednost $NL(v)(i)$, ki jo izračunamo kot uteženo povprečje vseh slikovnih točk:

$$NL(v)(i) \coloneqq \sum_{j \in I} w(i,j) v(j) \ ,$$

kjer je $\{w(i,j)\}_j$ množica uteži, ki zadošča pogojema $0 \leq w(i,j) \leq 1$ in $\sum_j w(i,j) = 1$.

Namesto, da bi primerjali le vrednosti slikovnih točk i in j, bomo utež $w(i,j)$ definirali kot merilo podobnosti okolic slikovnih točk i in j. Okolico slikovne točke k označimo z $\mathcal{N}_k$. Utež $w(i,j)$ bo odvisna od podobnosti vrednosti $v(\mathcal{N}_i)$ in $v(\mathcal{N}_j)$. Izračunali jo bomo z evklidsko razdaljo $\|v(\mathcal{N}_i) - v(\mathcal{N}_j)\|^2_{2,a}$, kjer je $a > 0$ standardni odklon Gaussovega jedra. Uporabo evklidske razdalje za izračun podobnosti utemeljimo z izračunom matematičnega upanja razdalje $\|v(\mathcal{N}_i) - v(\mathcal{N}_j)\|^2_{2,a}$, ki da rezultat

$$E[\|v(\mathcal{N}_i) - v(\mathcal{N}_j)\|^2_{2,a}] = \|u(\mathcal{N}_i) - u(\mathcal{N}_j)\|^2_{2,a} + 2\sigma^2.$$

Pri računanju podobnosti dveh območij s prisotnim šumom, bomo torej dobili enako stopnjo podobnosti, kot bi jo dobili v primeru, ko šum ni prisoten. S tem smo se izognili problemu, ki smo ga imeli pri sosedskem filtriranju.

Utež $w(i,j)$ izračunamo s formulo

$$w(i,j) = \frac{1}{Z(i)} e^{-\frac{\left\|v(\mathcal{N}_i) - v(\mathcal{N}_j)\right\|_{2,a}^2}{h^2}},$$

kjer je normalizacijska konstanta $Z(i)$ enaka

$$Z(i) = \sum_j e^{-\frac{\left\|v(\mathcal{N}_i) - v(\mathcal{N}_j)\right\|_{2,a}^2}{h^2}}.$$

Utež $w(i,j)$ bo torej poskrbela, da bomo pri izračunu nove vrednosti za slikovno točko i, povprečili le vrednosti tistih slikovnih točk, ki imajo podobno geometrijsko strukturo v okolici.

Primer 2.35

Pravilnost postopka

Če upoštevamo stacionarnost, potem algoritem NL-povprečenja konvergira k vrednosti pogojnega matematičnega upanja za piksel i. V tem primeru stacionarnost pomeni, da se z večanjem velikosti slike ohranijo podobna območja z detajli na sliki.

Naj bo V naključno polje in v njegova realizacija. Z Z označimo zaporedje $Z_i = \{X_i, Y_i\}$, kjer je $Y_i = V(i)$ realna vrednost in je $X_i = V(\mathcal{N}_i \setminus \{i\})$ vektor iz $\mathbb{R}^p$. NL-povprečenje vzamemo kot pogoj pri pogojnemu matematičnemu upanju, izračun je tak

$$r(i) = E[Y_i \mid X_i = v(\mathcal{N}_i \setminus \{i\})].$$

Izrek 2.36 *Naj bo $Z = \{V(i), V(\mathcal{N} \setminus \{i\})\}$ za $i = 1, 2 \ldots$ popolnoma stacionaren in mešan proces. Z NL_n označimo algoritem NL-povprečenja, ki ga uporabimo na zaporedju $Z_n = \{V(i), V(\mathcal{N}_i \setminus \{i\})\}_{i=1}^n$. Potem velja*

$$|N_n(j) - r(j)| \to 0$$

skoraj za vsak $j \in \{1, \ldots, n\}$.

Zgornji izrek nam pove, da algoritem NL-povprečenja popravi sliko s šumov in ne poskuša ločiti narazen šuma in originalne slike.

V tem primeru predpostavimo, da imamo aditiven bel šum. Naslednji rezultati nam povedo, da je pogojno matematično upanje funkcija $V(\mathcal{N}_i \setminus \{i\})$, ki minimizira povprečno kvadratno napako za pravo sliko (originalno).

Izrek 2.37 *Naj bodo V, U in N naključna polja, za katera velja, da je $V = U + N$, kjer je N neodvisno od signala prisoten bel šum. Tedaj veljata naslednji trditvi:*

1. *Za vsak $i \in I$ in $x \in \mathbb{R}^p$ velja $E[V(i) \mid X_i = x] = E[U(i) \mid X_i = x]$.*
2. *matematično upanje $E[U(i) \mid V(\mathcal{N}_i \setminus \{i\})]$ je funkcija v odvisnosti od $V(\mathcal{N}_i \setminus \{i\})$, ki minimizira kvadrat napake*

$$\min_g E[U(i) - g(V(\mathcal{N}_i \setminus \{i\}))]^2.$$

Časovna zahtevnost

Kljub temu, da nelokalni algoritem za odstranjevanje šuma, da zelo dobre rezultate in je robusten, pa je njegova uporaba zaradi visoke časovne zahtevnosti dokaj omejena.

Recimo, da je velikost slike n^2 in velikost okolice $\mathcal{N}_k$ enaka M^2 (M je konstanta). Časovna zahtevnost za izračun uteži $w(i,j)$ je enaka M^2. Izračun vrednosti $NL(v)(i)$ zahteva izračun n^2 uteži, kar pomeni, da je časovna zahtevnost izračuna ene vrednosti $NL(v)(i)$ enaka $\mathcal{O}(M^2 \cdot n^2)$. Skupna časovna zahtevnost je torej $\mathcal{O}(M^2 \cdot n^4)$, saj moramo izračunati vrednost $NL(v)(i)$ za vsako izmed n^2 slikovnih točk.

Prva izboljšava, ki jo naredimo v praksi za zmanjšanje časovne zahtevnosti, je omejitev območja pri računanju vrednosti $NL(v)(i)$. Namesto, da vsoto naredimo po vseh slikovnih točkah, jo naredimo le na omejeni okolici slikovne točke i. V članku izbrana velikost je 21×21. Skupaj z izbiro $M = 7$ dobimo izboljšano časovno zahtevnost $\mathcal{O}(49 \cdot 441 \cdot n^2)$. Pridobili smo kvadratno časovno zahtevnost, kar pa še vedno ni dobro za širšo uporabo nelokalnega filtra v praksi. Poleg tega omejitev območja prinese slabšo kakovost filtrirane slike.

Pohitritev algoritma

Wang je skupaj s sodelavci v članku [17] predstavil algoritem za nelokalno odstranjevanje šuma, ki v praksi porabi občutno manj časa za izračun kot originalni algoritem, pri čemer pa ohrani kakovost filtriranja. Kot smo videli, največ časa porabimo za izračun evklidske razdalje med dvema okolicama. Za to razdaljo bomo uvedli novo oznako $S(i,j)$:

$$S(i,j) = \|\mathcal{N}_i - \mathcal{N}_j\|^2 \tag{2.13}$$

$$= \sum_{l=0}^{M-1}\sum_{m=0}^{M-1} [v(\mathcal{N}_i)(l,m) - v(\mathcal{N}_j)(l,m)]^2 \, . \tag{2.14}$$

Če sliko I postavimo v koordinatni sistem in jo zrcalimo preko koordinatnega izhodišča (glej sliko **??**), lahko $\mathcal{N}_j(l,m)$ zapišemo kot $\mathcal{N}_j(l - x'_j, m - y'_j)$, kjer je $x'_j = \frac{3M}{2} + x_j$ in $y'_j = \frac{3M}{2} + y_j$. Enačba (2.13) se s tem prepiše v obliko:

$$\begin{aligned} S(i,j) &= \sum_{l=0}^{M-1}\sum_{m=0}^{M-1} [v(\mathcal{N}_i)(l,m) - v(\mathcal{N}_j)(l - x'_j, m - y'_j)]^2 \\ &= N_i^2 + N_j^2 - N_i * N_j \, , \end{aligned}$$

kjer so

$$N_i^2 = \sum_{l=0}^{M-1}\sum_{m=0}^{M-1} (v(\mathcal{N}_i)(l,m))^2,$$

$$N_j^2 = \sum_{l=0}^{M-1}\sum_{m=0}^{M-1} (v(\mathcal{N}_j)(l - x'_j, m - y'_j))^2,$$

$$N_i * N_j = 2 \cdot \sum_{l=0}^{M-1}\sum_{m=0}^{M-1} (v(\mathcal{N}_i)(l,m) \cdot v(\mathcal{N}_j)(l - x'_j, m - y'_j)) \, .$$

Vrednosti N_i^2 in N_j^2 lahko s pomočjo podatkovne strukture *vsote kvadratov* (*angl. Summed Square Image*), ki je predstavljena v 2.4.3, izračunamo brez uporabe množenj.

Vsota kvadratov

Za slikovno točko (x_0, y_0) naj bo

$$SSI[y_0, x_0] \coloneqq \sum_{x \leq x_0, y \leq y_0} v^2(y, x).$$

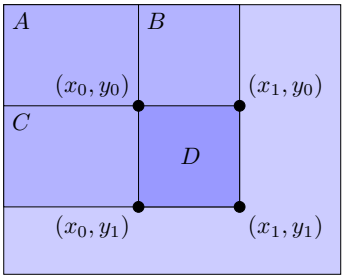
Algoritem 1 Izračun $n \times m$ matrike vsote kvadratov.

```
1: SSI ← Prazna n × m matrika.
2: SSI[0, 0] ← v²(0, 0)
3: for x₀ ∈ {1, ..., m − 1} do
4:     SSI[0, x₀] ← SSI[0, x₀ − 1] + v²(0, x₀)
5: end for
6: for y₀ ∈ {1, ..., n − 1} do
7:     SSI[y₀, 0] ← SSI[y₀ − 1, 0] + v²(y₀, 0)
8: end for
9: for x₀ ∈ {1, ..., m − 1} do
10:    for y₀ ∈ {1, ..., n − 1} do
11:        SSI[y₀, x₀] ← SSI[x₀ − 1, y₀] + SSI[x₀, y₀ − 1] − SSI[x₀ − 1, y₀ − 1] + v²(y₀, x₀)
12:    end for
13: end for
```

Zgornji algoritem ima časovno zahtevnost $O(nm)$, pri čemer je $n \times m$ velikost slike. Vsako slikovno točko po zgornjem algoritmu namreč obiščemo natanko enkrat.

Če nas sedaj zanima kolikšna je vsota kvadratov vrednosti slikovnih točk na pravokotnem imamo dano strukturo SSI, lahko vrednost poljubnega dela slike ali slikovne točke izračunamo v konstantnem času. Naj bo D območje za katerega želimo izračunati vsoto kvadratov vrednosti. Vrednost S_D določimo na naslednji način:

$$\begin{aligned} S_D &= S_{A \cup B \cup C \cup D} + S_A - S_{A \cup C} - S_{A \cup B} \\ &= SSI(x_1, y_1) + SSI(x_0, y_0) - SSI(x_0, y_1) - SSI(x_1, y_0). \end{aligned}$$



2.4.4 Implementacija in primeri

Poglavje 3

Generiranje šuma in teksture papirja

Poglavje 4

Likovno upodabljanje slik

Likovno upodabljanje slik (v nadaljevanju LUS) je področje v računalniški grafiki, ki se ukvarja z vprašanjem, kako realistični medij slike (lahko tudi video) pretvoriti v artistično likovno delo. V nasprotju z realističnim upodabljanjem, pri katerem je fokus na detajlih, pri LUS poskušamo upodobiti imanentne informacije. LUS ima zelo široko uporabo pri upodabljanju slik, risb, (tehničnih) ilustracijah, risanih animacijah ... V nasprotju s fotorealističnimi slikami imajo tako dobljena likovna dela namreč to prednost, da lahko likovni umetnik (ali tehnični risar) da poudarek na določen detajl slike, omeji količino podrobnosti, prilagodi paleto barv ali npr. doda sliki svojo osebno noto. Pri NPR-ju s pomočjo računalnika ustvarimo likovne umetnine, ki posnemajo določen likovni stil.

NPR se je razvilo iz področja prepoznave določenih oblik na sliki in digitalne obdelave slik. V prejšnjem poglavju smo spoznali filtre za iskanje robov na sliki, zamegljenje ... V prvem razdelku si bomo sedaj pogledali primer artističnega filtra.

4.1 Artistični filter

Primer.

Obstaja tudi vrsta komercialnih programov, ki ustvarjajo slike s pomočjo artističnih filtrov, npr. Adobe Photoshop.

4.2 Splošni pregled dela na področju LUS

V tem poglavju bomo naredili pregled dela na področju likovnega upodabljanja slik.

4.2.1 Ilustracije

Seligmann in Feiner sta opisala metodo za avtomatično konstruiranje ilustracij, s katerimi bi dosegli določeni cilj [31]. Njihov osnovni cilj je high-level of compositing the best model for comunicatting a particular intent. Yessios je opisal prototip sistema za upodabljanje materialov v arhitekturnih načrtih, kot so kamni, les, rože in tla. Namesto mehaničnega videza želijo ilustracijo prikazati z nežnejšimi potezami. Miyata [25] je opisal algoritem za upodabljanje vzorcev kamnitih sten. Appel je z soavtorji v [2] predstavil metodo, s katero narišemo črto, ki ima videz, da gre zadaj za drugo. Kamada in Kawai [16] sta posplošila nuno delo in ppredstavila različne parametre črte, kot so črtkanost, pikčastost,

. . . , s pomočjo katerih lahko skrite črte vizualno boljše predstavimo. Dooley in Cohen sta kasneje v [**?**, **?**] predstavila še več parametrov linij, kot je npr. debelina črte in podala diskusijo o načinu uporabnikove interakcije, da bi izboljašlai likovni vtis ilustracij, pri čemer bi obravnavali obliko objekta (linije, ki ga določajo) in njegovo senčenje.

Saito in Takahashi sta v [**?**] predstavila koncept *G*-bufferja za ustvarjanje razumljivih upodobitev 3D scen. Njihova metoda v bistvu po tem, ko ustvarijo množico *G*-bufferjev, uporablja tehnike iz procesiranja slik. V [**?**] sta Winkenbach in Salesin opisala tradicionalne pristope za upodabljanje slik na pen-and-ink način in pokazala, da lahko mnoge izmed njih implementiramo kot del avtomatskega likovnega upodabljanja slik. Predstavila sta stroke texture, s pomočjo katerega lahko tako teksturo kot tudi senčenje dosežemo z risanjem linij. Stroke texture dovoljuje tudi likovno upodabljanje slik, ki je odvisno od izbrane resolucije. Opisane tehnike sta demonstrirala s pomočjo kompleksnega arhitekturnega modela, ki vključuje tudi Lloyd's Wright's Robie House. Svoje delo sta v [37] nadgradila, kjer sta uvedla nove algoritme in tehnike za likovno upodabljanje parametrično podanih površin, ki imajo nepravilno obliko. Predstavila sta idejo hatchinga, kjer kontroliramo gostoto risanja linij, da bi dosegli želejeni ton senčenja, obliko in teksturo. Pokazala sta tudi kako lahko ravninski zemljevid, ki je glavna podatkovna struktura pri njunem upodabljanju ilustracij, konstruiramo iz parametrično podanih površin in ga uporabimo za rezanje linij in ustvarjanje linij, ki omejuje objekte. Za ukrivljene objekte sta predstavila tudi senčenje, ki posnema linijo ukrivljenega objekta, ki ga upodabljamo.

Lansdown in Schofield sta v [**?**] sta predstavila sistem Piranesi, ki uporablja tehnike likovnega upodabljanja za ustvarjanje ilustracij iz podanega 3D modela. Piranedsi uporablja standarne grafične plinovode, s pomočjo katerih ustvari 2D reliefno sliko. Uporabnik lahko nato posameznim območjim na sliki določi teksturo ali pa je ta določena avtomatsko.

Salisbury je skupaj s sodelavci v [30] predstavil interaktivni sistem za risanje ilustracij na podlagi črnobele vhodne slike, kjer linije na ilustraciji posnemajo obliko objektov na vhodni sliki. Uporabnik, preko novih tehnik za dodajanje vektorskega polja, določi orientacijo za vsako posamezno območje. Računalnik nariše linije, ki temeljijo na uporabnikovih primerih liniji in poskusi doseči iste tone na ilustracji kot so na vhodni sliki, s pomočjo novega algoritma, ki primerja zamegljeno verzijo trenutne ilustracije z vhodno sliko. S poravnanim vektorskim poljem z orientacijami površin objektov na sliki lahko uporabnik ustvari teksturo, ki ima videz, da je pritisnjena na površino, namesto da z njo le potemnimo neketere dele. To ima kot posledico bolj privlačne ilustracije, kot smo jih lahko poprej upodobili iz 2D slik.

Goodwin [33] s sodelavcama je predstavil pristop za določanje debeline črt v računalniško generiranih ilustracijah gladkih površin. Predpostavka, da se poudarjene črte rišejo na tistih območjih slike kjer je temnejše senčenje, jih je vodila do preproste formule za debelino linij, ki omejujejo objekte in tistih linij, ki sugestirajo detajle na objektih. Formula je odvisna od globine, ukrivljenosti in smeri svetlobe. Za upodabljanje linij uporabijo lokalno obliko in relacijo globine.

4.2.2 Risbe

Risbe so med osnovnimi likovnimii komunikacijami, ki človeku pomagajo v možganih reproducirati vhodno sliko risbe. Risanje s svinčnikom lahko razdelimo v dve skupini glede na vhodne podatke. te lahko namreč dobimo kot 2D informacijo ali pa kot 3D model. Glavna likovna stila, ki ju s svinčnikom poskušamo upodobiti sta skica ()in hatching (z ri-

sanjem na istem mestu v enakih ali različnih smereh poskušamo doseči določeni ton slike). Sousa in Buchanan [?] sta predstavili modele za grafit, papir za risanje, radirke. S temi modeli predstavi teksturo, ton in Modeli so narejeni na podlagi interakcije med grafitom svinčnika in papirja ter na podlagi lastnosti radiranja in smoofanja S pomočjo parametrov, kot je

Lee s sodelavci so v [?] iz podane 3D površine objekta izločili linije, ki objekt omejujejo in jih predelali za simulacijo ročno narisanih linij. Senčenje simulirajo s pomočjo teksture svinčnika in pa usmerjenih linij. Temelji na svetlobi in materijalu danega 3D modela. Pri upodabljanju slik s svinčnikom so algoritmi najpogosteje taki, da dobijo za vhodni podatek 3D model in nato poskušajo dobiti informacijo o linijah za risbo iz modela [?, ?, ?, ?]. Rusinkiewicz je v [?] naredil pregled likovnega upodabljanja s svinčnikom na podlagi danega 3D modela.

V [?] je bil predstavljen model senčenja. Različne hatching stile sta definirala Hertzmann in Zorin v [?]. Praun pa je skupaj sodelavci [?] razvil algoritem za hatching, ki se izvaja v realnem času. V [?] je predstavljen algoritem za risanje potretov s svinčnikom s stilom, ki posnema določeno tradicijo ali pa stil risarjev.

Mao s sodelavci v [?] so predstavili postopek za lokalno strukturo orientacije na podlagi vhodne slike in vključili linearno integracijsko konvolucijo (LIC) z namenom simulacije efekta senčenja. Yamamoto pa je s sodelavci v [?] je razdelil vhodno sliko na plasti z različnimi toni v različnih obsegih. Li in Huang [?] sta uporabila geometrijske parametre, from ...

4.2.3 Painterly rendering

Cohen [5, 26] se je problema risanja oz. slikanja lotil z vidika umetne inteligence. Ustvaril je sistem Aaron, ki na podlagi množice randomiziranih pravil ustvari originalno kompozicijo v določenem artističnem stilu. Aaron ima na voljo tudi robotsko slikanje.

Haeberli je v [9] predstavil interaktivni sistem za slikanje, ki ustvari impresionistično sliko iz fotografije. Poteze čopiča je upodobil v vrstnem redu. Vsaka poteza je imela parametre (položaj, barva, velikost in oblika), ki so bili večinoma določeni interaktivno, tj. s pomočjo uporabnika. Shiraishi in Yamaguchi [23] določata orientacijo in obliko potez čopiča na podlagi lokalnih značilnosti vhodne slike. Curtis in ostali avtorji so v [6] predstavili semi-avtomatski algoritem za upodabljanje watercolor slik. Njihov algoritem ne bo ustvaril vidnih potez čopiča, zato s tem izgubijo artistični vtis slikanja s čopičem. Litwinowicz [21] je opisal avtomatičen sistem za postavitev potez čopiča na podlagi vektorskega polja izračunanega iz gradienta vhodne slike in uporabil rezanje potez na robovih slike. Izhodna slika algoritma je naslikana v impresionističnem stilu, za katerega so značilne kratke poteze. Sistem je razširil na likovno upodabljanje posnetkov. Za upodabljanje posnetkov je uporabil kratke poteze in premikal narisane poteze na platnu s pomočjo izračunanega optičnega toka, ustvarjenega na podlagi posnetka. Pristop, ki ga je Litwinowicz uporabil za postavljanje potez na platno (tj. postavitev potez na mrežo) je najbolj pogost način, ki ga uporabljajo za upodabljanje slik s čopičem. Razlike med algoritmi so v obliki in orientaciji potez. Treavett in Chen [34] sta predstavila način za likovno upodabljanje s statistično analizo vhodne slike, na podlagi katere določita položaj, orientacijo in velikost upodobljenih potez. Litwinowicz in Treavett s Chenom so sliko pobarvali enoplastno. Pri enoplastnem slikanju je potrebno natančno predviditi položaje in obliko potez, kar pa je težaven problem. Pomankljivost takih algoritmov je torej v tem, da ne moremo morebi-

tnih detajlov in napakic, ki nastanejo pri slikanju ne moremo naknadno popraviti še z eno plastjo potez čopiča. Hertzmann je v [11] predlagal način upodabljanja slik z večplastnim slikanjem, kjer postavlja na platno poteze čopiča različnih debelin in oblik, ki jih določi s pomočjo Bezierovih krivulj. Začetno točko poteze ne postavi na mrežo slike, temveč v okolici mrežnih točk poišče točko z največjo napako. Kontrolne točke Bezierovih krivulj pa poišče na podlagi izračunanega gradienta vhodne slike. Tak način upodabljanja slikanja je tudi najbolj podobno dejanskemu načinu slikanja. Slikar namreč najprej naredi grobo sliko, ki jo nato z manjšimi čopiči popravi na delih, kjer je potrebno več poudarka in detajlov. S pomočjo te metode lahko ustvarimo sliko, ki ima artistični vtis, vendar pa je potratna in detajlov na sliki vedno ne nariše lepo. To metodo je Hertzmann skupaj s Perlinom razširil na upodabljanje posnetkov [1].

Ročno narisane poteze s čopičem nimajo konstantne barve vzdolž poteze, temveč imajo vgrajeno strukturo. Običajna metoda za upodobljanje potez čopiča je s pomočjo antialiazirane črte, ki ima konststno barvo. Litwinowicz je v [21] predstavil idejo dodajanja teksture in svetlobe pri upodabljanju potez. Hertzmann je v [13] predstavil posebno tehniko za dodajanje efekta svetljenja za likovno upodobljene slike s pomočjo višinskega polja slike in bump-mappinga. V [15] so Huang in sodelavci predstavili podoben način kot je bump-mapping, vendar je bistveno enostavnejši za implementacijo. Predstavili so model anizotropičnega čopiča. Upodobaljanje potez poteka tako, da pripravljeno masko čopiča (na podlagi ...)

Hays in Essa sta v [10] predstavila uniform likovno upodabljanje za slike in posnetke. Za razliko od večine prejšnjih algoritmov, so poteze pri tej metodi postavljene na večjem platnu. Orientacije za poteze so izračunane z RBF interpolacijo. Pri upodabljanju posnetkov, se parametri potez med postopkom spreminjajo, orientacija potez pa je določena z vektorskim poljem na podlagi gradienta posameznih sličic, različne frekvence robov na sliki pa pripomorejo h upodobitvi detajlov na sliki.

Uporaba informacije o 3D geometrijski strukturi slike, ki jo želimo likovno upodobiti, nam dovoljuje veliko več fleksibilnosti pri ustvarjanju posnetkov. Meier iz Walt Disney Feature Animation je v [24] predstavila avtomatičen sistem, ki postavlja poteze čopiča na podlagi opisa posameznih delov slike. V filmih *Tarzan* [7] in *What Dreams May Come* so bile uporabljene razširitve tega algoritma. Tak sistem naravno zagotavlja časovno koherenco in ekonomičnost likovnega upodabljanja posnetka. Vendar pa podatki o geometrijski strukturi niso vedno dosegljivi. Klein [18] je objekte upodobil s pomočjo filtrirane strukture. Uporaba 3D strukture tipično proizvede drugačen srtistični učinek, saj so poteze čopiča vezane na geometrijsko struktuo objektov. Turk in Banks sta v [35] za ilustriranje vektorskega polja z uporabila relaksacijo. Njuno delo in uporabo relaksacije, ki jo je uporabil Haeberli v [9], je Hertzmann razširil v članku [12].

Lee je v [19] predstavil novo metodo za za določanje oprientacije potez čopiča v likovnem upodabljanju posnetkov. Informacijo o premikih, ki jo dobimo iz zaporedja sličič posnetka uporabimo za orientiranje potez čopiča v območjih, kjer je bilo zaznano večje gibanje. V območjih z rahlio zaznanim gibanjem pa določimo orientacijo potez čopiča na podlagi gradienta slike. Določanje orientacije po tem postopku ima prednost pri izražanju gibanja objektov.

Poglavje 5

Likovno upodabljanje s potezami

Likovno upodabljanje s potezami (*kratica* LUP; *angl. stroke–based rendering*) je skupni pojem za likovna upodabljanja slik, pri katerih na risalno podlago na predpisan način postavljamo diskretne elemente. Diskretni elementi so lahko poteze čopiča, pike, kvadratki, mozaične ploščice ...

Cilj SBR je taka postavitev diskretnih elementov na risalno podlagi, ki bo zadostila našim pogojem in ciljem. Skupni cilj je, da s postavitvijo diskretnih elementov na risalno podlago skušamo ustavariti sliko, ki bo po stilu čimbolj podobna nekemu stilu oz. bo nosila določen slikarski stil. Po drugi strani želimo, da naša slika ne deluje preveč umetno, potrebno je tudi omejeiti na nek način število diskretnih elementov, njihovo obliko, barvo, ...

Algoritem v splošnem poteka v več fazah:

1. postavitev diskretnih elemntov na risalno podlago;
2. rendiranje diskretnih elementov.

Najprej definirajmo pojma strokesa in podatkovno strukturo slika.

Definicija 5.1 Strokes je podatkovna struktura, ki jo lahko rendiramo. Model strokesa je opis strokesa, ki nosi podatke o obliki, poziciji in parametrih za njegovo rendiranje.

Strokesi, ki jih bomo uporabljali mi, bodo več vrst:

- pike (model pike: radij, center, barva);
- poteze s čopičem (enostavnejši in bolj bogat model - opisa sledita kasneje);
- ploščice, ki potrenujemo pri mozaikih;
- trganke (trganke iz revije).

Definicija 5.2 Struktura slike je podatkovna struktura, ki vsebuje:

- Platno, na katerega bomo slikali (barva platna in njegova struktura);
- Urejen seznam strokesov, ki jih določajo parametri.

Sedaj potrebujemo še oceno, ki nam bo povedala kako dobra je struktura slike, glede na cilje, ki smo si jh zadali.

Definicija 5.3 SBR energijska funkcija je funkcija $E : I \to R$, pri čemer je I množica vseh možnih struktur slike in R množica relanih vrednosti.

Intuitivno lahko gledamo na energijsko funkcijo kot na merilo, ki nam pove, kako dobro narisana je slika. Cilj SBR-ja je naslikati sliko, ki bo imela čimmanjšo možno vrednost energije. Naš cilj pri risanju je, da bi dano sliko čimbolje narisali, torej da bo naša končna slika čimboljši približek originalne slike. To je enostavno doseči tako, da na risalno podlago naslikamo čimveč majhnih strokesov. Na ta način bomo dobili zelo dobro ujemenje. Na ta način bomo torej dobili zelo dober približek originalne slike. Ne bomo pa dosegli umetniškega cilja. Namreč, da bi poustavrili določen slikarski stil. Zato poleg zahteve, da dobimo čimboljšo ujemanje slike, dodamo zraven motnje, omejimo število strokesov, njihovo velikost itd.

Z energijsko funkcijo bomo torej določili slikarski stil. Slikarski stil lahko definiramo kot sledi:

Definicija 5.4 SBR stil je model strokesa skupaj z energijsko funkcijo (vključujoč vrednosti parametrov) preko strukture slike.

Z drugimi besedami, SBR algoritem ustvari sliko z določenim stilom tako, da minimizira določeno energijsko funkcijo.

Glede na postavljanje strokesov na risalno podlago, ločimo dve glavni vrsti SBR algoritmov:

1. optimizacijski algoritmi;

2. požrešni algoritmi.

V nadaljevanju si bomo pogledali algoritme iz obeh sklopov. Delo je povzeto po doktorski disertaciji od strica Hertzmanna.

5.1 Optimizacijski algoritmi

Pri požrešni metodi smo poteze postavljali na platno po vnaprej določenem pravilu. Ko smo potezo enkrat postavili na platno, je nismo več spreminjali. Opazimo pa, da bi bilo večkrat za dosego lepše slike bolje, da bi neki potezi spremenili obliko in barvo ali pa jo celo odstranili. Tak način razmišljanja uporablja optimizacijski pristop k likovnemu upodabljanju s potezami, ki ga bomo spoznali v tem razdelku.

Najprej moramo uvesti matematično merilo, ki bo za dano trenutno postavitev potez na platnu J izmerilo njeno kvaliteto.

Definicija 5.5 *LUP energijska funkcija* je funkcija $E : \mathcal{J} \to \mathbb{R}$, kjer je $\mathcal{J}$ množica vseh možnih slik.

Glavni dve merili, ki ju bomo upoštevali pri računanju vrednosti energijske funkcije, sta število potez in ujemanje slike J z vhodno sliko I. Seveda bi lahko z ogromnim številom majhnih potez zelo dobro aproksimirali vhodno sliko I, vendar pa ne bi dobili likovno zadovoljivega rezultata. Pri optimiziranju skušamo doseči, da neko sliko aproksimiramo z

manjšim številom potez. Na ta način v sliko vnesemo abstrakcijo in željeni likovni pridih. Osnovna energijska funkcija, ki jo bomo optimizirali, bo:

$$\begin{aligned} E(J) &= E_p(J) + w_n \cdot E_n(J), \\ E_p(J) &= \sum_{(x,y)\in J} \|J(x,y) - I(x,y)\|^2, \\ E_n(J) &= \text{število potez na sliki } J. \end{aligned}$$

S parametrom w_n nadziramo abstrakcijo slike. Uporaba majhne vrednosti parametra w_n pomeni, da bomo pri slikanju uporabili veliko število potez, posledično bomo dobili likovna dela, ki bodo verjetno bolj ali manj podobna začetni vhodni sliki I. Za dosego bolj likovno estetskega rezultata bomo torej običajno uporabljali večje vrednosti parametra w_n, saj bomo na ta način zmanjšali število končnih potez.

Izbira energijske funkcije, ki jo bomo uporabili pri optimiziranju, skupaj z modelom poteze določa *stil slike*. Določen stil slike dobimo torej tako, da minimiziramo energijsko funkcijo.

Opomba 5.6 Energijsko funkcijo lahko izberemo tudi kot merilo za kvaliteto sliko, ki jo dobimo pri slikanju s požrešno metodo.

Kljub temu, da bomo pri slikanju na optimizacijski način dobili veliko boljši rezultat kot pri požrešni metodi, pa tak način slikanja ni naraven. Slikar namreč na platnu ponavadi neke poteze, ko jo naslika na platno, običajno ne more več spreminati. Minus je tudi ta, da bo čas likovnega upodabljanja v nekaterih primerih drastično narastel (če bi pri požrešni metodi porabili pet minut, se kaj lahko zgodi, da bomo pri optimiziranju za dosego podobnega stila pri isti vhodni sliki potrebovalii pet ur). Optimizacijski pristop po drugi strani pri svojem ustvarjanju uporabljajo sestavljalci mozaiki. Sestavljalec najprej na podlagi s prestavljanjem, izbiro ploščiš druge barve in spreminjanjem oblike ploščic sestavi željeno sliko, nato pa ploščice zacementira.

V podrazdelkih, ki sledita, bomo opisali dva optimizacijska pristopa. Prva skupina algoritmov, ki jo bomo opisali, so Voronojevi algoritmi. Z njimi bomo pokazali kako lahko sestavimo mozaik. Nato bomo spoznali še izkustvene algoritme, pri katerih bomo uporabljali hevristično izbrane teste, s pomočjo katerih bomo minimizirali energijsko funkcijo. Voronojevi algoritmi imajo boljšo časovno zahtevnost, ampak je ne moremo prilagoditi različnim vrstam problemov. Medtem ko imajo izkustveni algoritmi slabšo časovno zahtevnost, pa jih lahko z izbiro energijske funkcije in parametrov prilagodimo različnim problemov in poustavrimo veliko več likovnih stilov.

5.1.1 Voronojevi algoritmi

Pri razvoju Voronjevega algoritma za LUP, si bomo pomagali z znanjem računske geometrije. V primeru, ko bomo na podlago postavljali neprekrivajoče se poteze s podobno obliko, bomo videli, da lahko problem prevedemo na znan Voronojev diagram. Metoda, ki jo bomo predstavili v našem delu, je bila preedstavljena v članku [**?**].

Lloydova metoda

Množico enakomerno razporejenih diskretnih elementov na risalni podlagi bomo določili s pomočjo iterativnega optimizacijskega postopka, ki bo minimiziral predpisano energijsko

funkcijo. Naj bo $p = (x, y)$ slikovna točka in označimo s C_i posebno slikovno točko, ki ji pravimo *center*. Diskretne elemente bomo postavili na sliko tako, da bodo imeli središča v centrih. Naj bo $L_p^i \in \{0, 1\}$ označitev slikovnih točk, pri čemer $L_p^i = 1$ pomeni, da slikovna točka p pripada centru C_i. Vsaka slikovna točka pripada natanko enemu centru, veljati mora $\sum_p L_p^i = 1$.

Naloga: Izbira množice centrov in označitve slikovnih pik, ki minimizira naslednjo energijsko funkcijo [1]:

$$\begin{aligned} E(I) &= \sum_{p \in I} L_p^i \left\| p - C_i \right\|^2 \\ &= \sum_{p \in I} L_p^i ((p_x - C_{i,x})^2 + (p_y - C_{i,y})^2). \end{aligned}$$

Z drugimi besedami, vsaki slikovni točki želimo pripisati center, ki ji je najbližji. Če bi poznali množico centrov vnaprej, bi optimalno označitev enostavno izračunali tako, da bi vsaki slikovni točki pripisali njej najbližji center. Slednjo nalogo poznamo pod imenom Voronojev diagram. Več o Voronojevem diagramu si lahko preberemu v [**?**].

Kot je razvidno iz slike **??**, naključna izbira centrov ne bo dala zadovoljivega rezultata. Izboljšava, ki jo lahko naredimo je ta, da spremenimo položaje centrov. Energijsko funkcijo želimo optimizirati samo glede na množico centrov, zato moramo rešiti enačbo $\frac{\partial E(I)}{\partial C_i} = 0$. Nove položaje centrov izračunamo torej po formuli

$$C_i = \frac{\sum_p L_p^i p}{\sum_p L_p^i}.$$

Zgornja formula nam da srednjo vrednost vseh položajev slikovnih točk, ki smo jih na začetku pripisali določenemu centru. Zatem, ko spremenimo množico centrov, zopet na novo izračunamo označitev. Ponavljanje zgoraj opisanih korakov poznamo pod imenom Lloydova metoda. Psevdokoda Lloydeve metode je predstavljena v algoritmu **??**. Na sliki **??** je prikazana množica centrov in Voronojev diagram po tem, ko na prvotni množici centrov uporabimo Lloydovo metodo. Pravimo, da zgornji algoritem konvergira, ko se vrednost energijske funkcije med posameznima korakoma optimizacije ničveč ne spreminja. na vsakem koraku se bo vrednost energijske funkcije kvečjemu zmanjšala, saj na vsakem koraku energijsko funkcijo minimiziramo glede na določene parametre. Metoda bo zagotova konvergirala, saj je na voljo le končno mnogo kombinacij množic centrov in označitev slikovnih pik, na vsakem koraku pa se vrednost energijske funkcije kvečjemu zmanjša.

Prilagojena Lloydova metoda za reševanje problema SBR

Sedaj, ko imamo na voljo algoritem, ki nam enakomerno razporedi centre na sliki, lahko Lloydovo metodo prilagodimo tako, da jo bomo uporabili za rešitev problema SBR. Najosnovnejši problem SBR, ki ga obravnavamo je taka postavitev točk na sliko, ki bo čimbolje aproksimirala dano črnobelo sliko. Za ta problem obstajajo že rešitev Poglejmo si rešitev,

[1] Pri NPR-ju velikokrat namesto diskretne verzije izberejo zvezno verzijo problema. Ker pa je diskretni problem bližje realnemu problemu in za ta primer lahko pokažemo konvergenco algoritma, bomo mi izbrali diskretno verzijo problema.

ki jo je predstavil **??**. Namesto, da spreminjamo velikost narisanih točk, spreminjamo njihovo gostost. Uporabili bomo funkcijo ploskovne gostote $\rho(p)$, ki bo določala gostost točk na posameznih območjih vhodne slike. Funkcijo ploskovne gostote izpeljemo iz vhodne slike $T(p)$, ki smo jo predhodno pretvorili v črnobelo, po formuli

$$\rho(p) = 1 - \frac{T(p)}{m},$$

kjer je m največja vrednost slike T. Nova energijska funkcija za ta problem se glasi

$$\begin{aligned} E(I) &= \sum_{p \in I} L_p^i \rho(p) \left\| p - C_i \right\|^2 \\ &= \sum_{p \in I} L_p^i \rho(p)((p_x - C_{i,x})^2 + (p_y - C_{i,y})^2). \end{aligned}$$

Podoben razmislek kot pri navadni Lloydovi metodi nas pripelje sedaj do nekoliko spremenjene Lloydove metode. Postopek označitve slikovnih točk ostane isti kot prej. Ker smo vsaki točki s funkcijo ploskovne gostote pripisali utež, moramo pri računanju novih centrov te uteži upoštevati. Namesto običajne srednje vrednosti, bomo torej morali za posamezne centre vzeti uteženo povprečje slikovnih točk iz pripadojoče celice. Nove centre izračunamo po formuli

$$C_i = \frac{\sum_p L_p^i \rho(p) p}{\sum_p L_p^i \rho(p)}.$$

Računanje novih centrov lahko pohitrimo tako, da vsote uteži, ki nastopajo v imenovalcih, izračunamo vnaprej. S to metodo dobimo ostrejše robove na sliki, saj je gostota točk odvisna od tonske slike.

Lloydovo metodo pa lahko uporabimo tudi za sestavljanje mozaikov na podlagi dane barvne vhodne slike. Preprosti pristop, ki ga lahko uporabimo pri mozaiku je, da izračunamo Voronojev diagram in za barve posameznih celic uporabimo barvo, ki jo ima center na vhodni sliki. Ta pristop ni najbolšji, saj pri njem dobimo mozaik, ki ima preveč nepravilne oblike ploščic.

Hausner **??** opisuje dve izpopolnitvi metode. Najprej naodmestimo evklidsko razdaljo oz. L_2 normo z L_1 normo ($\|v\|_1 = |v_x| + |v_y|$). Druga izboljšava je določitev vektorskega polja $\Phi(p)$ sliki, da dobimo konsistentno orientacijo ploščic pri končnem mozaiku. Na ta način ploščicam v mozaiku pripišemo smer, $\phi_i = \Phi(C_i)$. Nova energijska funkcija je enaka

$$E(I) = \sum_{p \in I} L_p^i \left\| R_{\Phi(C_i)}(p - C_i) \right\|_1^2,$$

kjer je $R_{\Phi(C_i)}$ rotacijska matrika s smerjo $\Phi(C_i)$. Psevdokoda novega algoritma je povzeta v **??**. Zgornji algoritem ni več optimizacijski, kajti posamezen korak pri optimizaciji ne bo dal nujno boljše rešitve. Še več, zgornji algoritem pri postavitvi ploščic ne upošteva barve ploščic, temveč barvo ploščicam priredi šele, ko so znani končni položaji ploščic. Čeprav teoretično izgleda, da algoritem ni najboljši, pa se v praksi izkaže, da z njim dobimo zadovoljivo dobre rezultate.

Tlakovanje lahko prilagodimo tako, da ročno razdelimo območja slike in posamezna območja tlakujemo neodvisno (izberemo lahko različne parametre, npr. velikosti ploščic). Iz prvotne slike lahko odstranimo tudi robove, na ta način se izognemo temu, da bi ploščica imela center na katerem izmed robov, kajti to poslabšuje kvaliteto mozaika.

5.1.2 Izkustveni algoritmi

Voronojev algoritem se ne da lahko razširiti tako, da bi upošteval tudi barvo ploščic in obravnaval dejstvo, da se lahko diskretni elementi tudi prekrivajo. Trenutno so edini algoritmi, ki znajo upoštevati tudi to, *izkustveni algoritmi*. Izkustevni algoritem lahko uporabimo za reševanje kateregakoli SBR problema.

Ideja: Na vsakem algoritmu je predlagana nova sprememba za že postavljeno strukturo. V primeru, ko je sprememeba takšna, da zmanjša vrednost energijski funkciji, spremembo izvedemo, sicer jo zavržemo.

Če je sistem, ki predlaga na vsakem koraku algoritma novo spremembo, dobro zasnovan, potem bo algoritem konvergiral k rešitvi, ki bo imela nizko vrednost energijske funkcije. Zagotovila, da bomo prišli do take rešitve, pa pri tem novem algoritmu nimamo. Prav tako nimamo zagotovila, da bomo prišli do primerne rešitve v nekem zadovoljvio majhnem času. V algoritmu **??** je povzeta psevdokoda za izkustveni algoritem. V zgornjem algoritmu pri zanki nismo uporabili določenega zaustavitvenega pogoja, namreč pustimo, da zaustavitveni pogoj določi uporabnik. Glavna naloga pri tem algoritmu je, da postavimo dober sistem za predlaganje sprememb. Naključna izbira novih sprememb bi povzročila trošenje časa za to, da bi sistem predlagal rešitve, ki ne bi ničesar doprinesle k boljšemu rezultatu. Zato je bolje, da vzpostavimo polavtomatičen sistem, v katerem lahko uporabnik predlaga nove spremembe. Izkustevni algoritmu so v resnici podobni požrešnim algoritmom, pri obojih namreč za boljši rezultat uporabimo polavtomatiziran sistem. Kakorkoli, pri izkustvenem algoritmu se bo sprememba zgodila le, če bo nova sprememba doprinesla k boljšemu rezultatu. Pri izkustvenem algoritmu torej ni potrebno poskrbeti za to, da algoritem avtomatično ustvari nov diskretni element, ki bo pripomogel k boljšemu mozaiku.

Haeberli **??** je predstavil prvi izkustveni algoritem za likovno upodabljanje slik. Na vsakem koraku je izvedel spremembe določenega števila diskretnih elementov, izvedene spremembe nato ohrani, če je vsota kvadratov razlik glede na vhodno sliko zmanjšana.

Slikovno renderiranje z izkustvenim algoritmom

Heartzman je izkustveni algoritem uporabil za slikovno renderiranje. Naloga je poiskati sliko, ki se bo čimbolje ujemala z vhodno sliko, bo v celoti pokrita z barvo in bo uporabljala čimmanj potez. Vsaka poteza je definirana z debelino čopiča in seznamom kontrolnih točk. Energijska funkcija, ki jo bomo minimizirali je dana z

$$\begin{aligned} E(I) &= E_{app}(I) + E_{nstr}(I) + E_{cov}(I), \\ E_{app}(I) &= \sum p \in I w_{app}(p) \left\| I(p) - S(p) \right\|, \\ E_{nstr}(I) &= w_{nstr} \cdot (\text{število potez v } I), \\ E_{cov}(I) &= w_{cov} \cdot (\text{število nepobarvanih slikovnih točk v } I). \end{aligned}$$

Zgornja energijska funkcija je linearna kombinacija treh drugih energijskih funkcij, ki uravnavajo posamezne zahteve. Prva funkcija E_{app} nam pove koliko nova slika aproksimira vhodno sliko S. Število potez v drugi funkciji E_{nstr} poskrbi, da število potez na sliki ni preveliko, zahteva je namreč čimmanjše število potez. Tretja funkcija E_{cov} pa poskrbi za to, da nimamo na sliki nepobarvanih slikovnih točk. Uteži w_{app}, w_{nstr} in w_{cov} so parametri, ki jih lahko določi uporabnik. Uteži w_{app} uporabnik določi tako, da predlaga uteženo sliko, iz katere preberemo vrednosti uteži za posamezno slikovno točko.

Prvi dve funkciji uravnavata dve med sabo tekmovalni zahtevi: prva želi, da naredimo čimboljšo aproksimacijo prvotne slike, druga pa želi uporabiti čimmanj potez in s tem tudi čimamnj barve. Z uravnavanjem teh dveh parametrov lahko uporabnik doseže različne slikarske stile.

V primeru, ko uporabnik ne predlaga uteženе slike, bo algoritem za uteži w_{app} izbral vrednosti, ki jih bo prebral iz binarne slike, ustvarjene s pomočjo Sobelovega filtra. ta način da poseben poudarek na robove slik, kljub temu, da dobimo zadovoljive rezultate tudi, če bi uporabili kar konstantno utež skozi celotno sliko. Če dovolimo, da se vrednost parametra w_{app} spreminja skozi sliko, potem dobimo efekt, ki je podoben efektu, da imamo na različnih območjih slike drugačno energijsko funkcijo. Slika uteži $w_{app}(p)$ nam dovoljuje, da nadzorujemo koliko detajlov želimo na posameznih delih slike.

Relaksacija poteze: Osnovna strategija, ki jo uporabimo za sistem predlaganja novih sprememb, je uporaba *relaksacija poteze*. Gre za adaptacijo snakes **??** na energijsko funkcijo: poišče aproksimacijo lokalnega minimuma za dani položaj poteze.

High-level algoritem za modifikacijo poteze je:

1. Izračunamo energijo slike.
2. Izberemo eno izmed obstoječih potez.
3. Po potrebi spremenimo parametre poteze.
4. Relaksacija poteze.
5. Izračunamo novo energijo slike.

Podrobnosti implementacije so v poglavju 8.1. Na tem mestu omenimo naslednja dva aspekta implementacije:

- Pri postopku iskanja poteze izračunamo energijo slike brez te poteze. Na ta način je lahko predlagana sprememba brisanje poteze brez dodatnega stroška.
- Postopek relaksacije bi bil zelo drag postopek in težek za implementacijo, če bi ga naredili eksaktno. Namesto tega uporabimo aproksimacijo za računanje energije zunaj relaksacijske zanke in izračunamo eksatno vrednost energije, ko je sprememba predlagana.

Opis individualnih postopkov za generiranje spremembe:

- Dodajanje nove poteze: na podlagi izbrane točke na sliki ustvarimo novo potezo. Novo potezo vedno dodamo na konec seznama vseh potez (poteze narišemo na platno v vrstenm redu). Na način, ki je opisan v **??** dodamo nove kontrolne točke. Novo potezo nato relaksiramo. Rezultat tega iskanja je nato predlog za novo spremembo na sliki.
- Reaktivacija poteze: Dana poteza je relaksirana in sprememeba je nov predlog. Vendar pa v primeru, ko bi brisanje poteze iz slike energijo slike zmanjšajo bolj kot modifikacija poteze, potem potezo pobrišemo. Brisanje ene poteze ne vpliva na vrstni red potez. Na ta način bomo v postopku pobrisali poteze, ki so zadaj za ostalimi potezami, če za vključitev poteze na sliko obstaja kazen (tj. $w_{area} > 0$ ali $w_{nstr} > 0$).

- Odebelitev poteze: Če je radij poteze pod maksimumom, potem povečamo njen radij. Potezo zatem reaktiviramo. To potezo sedaj damo kot nov predlog za spremembo.
- Zoožanje poteze: Če je radij poteze nad minimumom, potem radij poteze zmanjašamo in potezo reaktiviramo.
- Prebaravanje: Barvo poteze nastavimo na povprečno vrednost barv območja vhodne slike, ki ga obsega poteza. Potezo nato reaktiviramo.

Kombinacija korakov: Individualne spremembe kombiniramo v zanke, da zagotovimo, da je vsaka poteza vsaj enkrat relaksirana.

- Postavitvena plast: Zanka preko slike, znotraj katere s pomočjo postopka *Dodajanje poteze* dodamo novo potezo s predpisanim radijem in naključno spremenjeno začetno točko.
- Splošna reaktivacija: Iteracija preko množice vseh potez v njihovem zaporedju in njihova *Relaksacija*.
- Odebeljenje/Zoožanje vseh potez: Za vsako potezo izvedemo postopek *Odebelitve* dokler sprememba ni zavržena li pa poteza pobrisana. V primeru, ko poteza ostane nespremenjena izvedemo postopek *Zoožitve* dokler ta sprememba ni zavrnjena lai pa poteza pobrisana.
- Splošno prebarvanje: Iteriramo preko množice potez in uporabimo postopek *prebarvanje*.
- Skripta: Izvedi zaporedje relaksacijskih zank. Če želimo narediti sliko na novo, uporabimo naslednji algoritem:

Običajno je $N = 2$. Zgornja skripta običajno pripelje do konvergence. Opazimo, da relaksacijsko zanko izvedemo na vseh potezah, ne le teh, ki imajo trenutno velikost.

5.2 Požrešne metode

Požrešne metode so najbolj običajni algoritmi s katerimi slikamo. Strokese dodamo na strukturo slike in jih po tem ne spreminjamo več, kot smo to storili pri optimizacijskih algoritmih. Pri tej vrsti lagoritma nimamo energijske funkcije, ki bi dolačala stil slike, temveč moramo na pravi način postaviti strokese na strukturo slike. Seveda lahko zapišemo energijsko funkcijo, ki je naravna.

Dve glavni vprašanji, ki ju imamo pri požrešni metodi slikanja, sta:

1. Kam postavimo strokese?
2. Kakšno obliko in barvo bodo imeli strokesi?

Pogledali si bomo različne vrste postavljanja strokesov na platno:

1. ravni strokesi (enoplastno, večplastno);
2. Bezierovi strokesi.

5.2.1 Slikanje z ravnimi potezami

Generiranje potez

Privzemimo, da je vsaka poteza čopiča renderirana z antialiazirano črto s centrom v (cx, cy), dano dolžino l, debelino čopiča R in orientacijo θ. Privzemimo, da so poteze čopiča generirane s centri v (cx, cy) na mreži slike, ki ima v x in y smeri stolpce in vrstice narazen za dve slikovni točki. Na ta način zagotovimo, da bo celotno platno prekrito z barvo oz. potezami čopiča. V praksi uporabniku dovolimo, da nastavi razdalje. Na začetku bomo privzeli, da je orientacija konstantna, izbrali bomo 45∘. Barva poteze je izbrana tako, da bilinearno interpoliramo barvno na vhodni sliki na točko (cx, cy) (imamo bravno lestvico $[0, 255]$). Poteze čopiča narišemo v naključnem vrstnem redu, da se izogmnemo regularnosti in pridobimo večji občutek naravnega procesa slikanja.

Naključne perturbacije

Dodajanje naključnih perturbacij k parametrom poteze, pomaga pri občutku, da je poteza narisana ročno. Naključnost dolžine in debeline poteze čopiča dosežemo s tem, da uporabnik za dolžino in debelino poteze čopiča predpiše interval dovoljenih vrednosti. barvo perturbiramo tako, da komponentam RGB dodamo vrednosti $\Delta r, \Delta g$ in Δb iz intervala $[-15, 15]$. Perturbirano barvo nato skaliramo z naključno vrednostjo $\Delta inteziteta$ iz intervala $[0.85, 1.15]$. Potem, ko dobimo končno perturbirano barvo, jo preslikamo nazaj na interval $[0, 255]$. Perturbiramo tudi orientacijo poteze, tako da orientaciji dodamo naključno vrednost iz intervala $[-15∘, 15∘]$.

Opomba 5.7 Dovoljene vrednosti za perturbacijo so izbrane na podlagi empiričnih poskusov.

Rezanje in upodabljanje potez čopiča

Potezo čopiča upodobimo tako, da narišemo antialiazirano črto skozi predpisan center v določeni orientaciji. Ker želimo ohraniti detajle in oblike na vhodni sliki, poteze čopiča porežemo pri sekanju z robovi na vhodni sliki. Na ta način se robovi na vhodni sliki bolj ali manj ohranijo. To dosežemo tako, da pričnemo v centru poteze in potezo rišemo toliko časa, dokler ta ne seka roba na vhodni sliki.

Algoritem za opis rezanja potez čopiča in njihovega upodabljanja:

1. Iz prvotne vhodne slike izračunamo matriko, v kateri za posamezno slikovno točko hranimo njeno inteziteto. Inteziteto za slikovno točko, ki ima komponente RGB na intervalu $[0, 255]$, izračunamo njeno inteziteto s formulo: $\frac{30 \cdot r + 59 \cdot g + 11 \cdot b}{100}$.

2. Dobljeno sliko I sedaj zameglimo z uporabo Gaussovega filtra. S tem filtrom na sliki zmanjšamo šum. Večji filter bo odstranil šum, vendar pa bo pri tem zameglil tudi detajle in robove na vhodni sliki. Manjši filtri pa bodo detajle in robove bolje ohranili, vendar ne bomo na ta način odstranili neželjenega šuma na sliki. Namesto Gaussovega filtra, lahko uporabimo aproksimacijo z B-zlepki. Uporabniku dovolimo, da nastavi ustrezne parametre.

3. Dobljeno zamegljeno sliko sedaj filtriramo s Sobelovim filtrom. Za vsako slikovno točko izračunamo gradient (G_x, G_y) in dobimo vrednost Sobelovega filtra za to slikovno točko kot $Sobel(x, y) = Magnitude(G_x, G_y)$. Dobimo novo sliko S.

4. Za dani center (c_x, c_y), orientacijo poteze θ in filtrirane slike S, moramo sedaj izračunati še začetno in končno točko poteze ((x_1, y_1) in (x_2, y_2)). Postopek pričnemo v centru (c_x, c_y) in rišemo potezo v določeni smeri, dokler ne dosežemo največje dovoljene dolžine ali pa sekamo rob na sliki S. Rob na sliki je bil najden, če vrednost graideinta oz. vrednost na sliki S pade v smeri risanja poteze. V algoritmu 2 je prikazana psevdokoda za postopek iskanja začetne in končne točke poteze.

Algoritem 2 Iskanje začetne in končne točke poteze. Center poteze je dan s točko (c_x, c_y), smer pa je dana z d_x, d_y. Izhodni podatki algoritma sta začetna in končna točka ((x_1, y_1) in (x_2, y_2)). Filtrirano sliko S pregledamo v enotskih korakih dokler ne sekamo roba ali dosežemo največje dolžine poteze. Spodnji postopek je prikaz za iskanje enega konca poteze. Za vsak konec namreč moramo iti v nasprotnih smereh pri risanju. Za iskanje točke (x_2, y_2) moramo tako nastaviti smer na $(d_x, d_y) = (-d_x, -d_y)$.

1: $(x_i, y_i) \leftarrow (c_x, c_y)$
2: $vred \leftarrow$ bilinearna vrednost filtrirane vrednosti slike S v točki (x_i, y_i)
3: $(x_t, y_t) \leftarrow (x_i + d_x, y_i + d_y)$
4: **if** $d((x_i, y_i), (x_t, y_t)) > \frac{l}{2}$ **then**
5: *stop*
6: **end if**
7: $vred_n \leftarrow$ bilinearna vrednost filtrirane vrednosti slike S v točki (x_t, y_t)
8: **if** $vred_n < vred$ **then**
9: *stop*
10: **end if**
11: $(x_i, y_i) \leftarrow x_t, y_t$
12: $vred \leftarrow vred_n$
13: *goto* 3

5. Potezo čopiča upodobimo z izračunano začetno in končno točko. Za barvo poteze izberemo kar barvo slikovne točke s koordinatami (c_x, c_y) na vhodni sliki. Perturbirati moramo barvo, parametre za perturbacijo imamo shranjene. Poteze upodobimo z antialiazirano črto, i ima okrog območje linearnega padca debeline slikovne pike in spremeni prosojnost od 1 do 0. Pri upodabljanju poteze poskrbimo tudi za to, da narisana črta ni čisto točno odrezana pri koncu, temveč gre nekoliko preko robnih točk. Na ta nain dosežemo boljši učinek ročno narisane črte. Na koncih črte lahko po želji narišemo tudi krogec, da dobimo črto, ki je na koncih zaobljena ali pa pustimo ravno odrezano črto.

Tekstura čopiča

Poteze čopiča lahko upodabljamo tudi s pomočjo teksture, tj. vzorca, ki vsebuje informacije o komponentah RGB in prosojnosti (alfa komponenta). Predpišemo padec, ki je odvisen od padca pri risanju antialiazirane črte, in ustvarimo pravokotnik, ki pokriva antialiazirano

črto. Teksturo poteze čopiča preslikamo na ta pravokotnik. Posamezne barve na potezi čopča pomnožimo s temi barvami, ki so na teksturi poteze čopiča.

Usmerjenost potez čopiča

Namesto predpisane orientacije poteze bomo izračunali orientacijo, ki bo sovpadala z linijami na slikami. Poteze tako rišemo v smeri konstantne ali skoraj konstantne barve. To orientacijo lahko aproksimiramo tako, da rišemo poteze v smeri normale na gradient slike I. Namreč, gradient nam pove v kateri smeri se slika najbolj spremeni, normala na gradient je torej smer v kateri se ne spremeni slika. Upoštevajoč to informacijo, pričakujemo, da lahko sliko lokalno avtomatično aproksimiramo z realtivno kratkimi potezami konstantne barve v smeri normale na gradient.

Eksperimentalno se izkaže, da uporaba istega Gaussovega filtra za detekcijo robov na sliki in zamegljenje slike za računanje gradienta ni optimalno. Izračunani gradient namreč ni bil dovolj gladek. Za izračun orientacije zato uporabimo večji Gaussov filter: uporabili so Gaussov filter, ki je bil za 4 slikovne pike večji od filtra za določanje robov.

Kakorkoli, v primeru, ko je magnituda gradienta blizu 0, se ne moremo zanesti na to, da bi bila določena smer uporabna. Zato uporabimo novo tehniko, ki spremeni gradientno polje tako, da poteze čopiča v območjih slike s konstantno ali skoraj konstsntno barvo gladko interpolirajo smeri, ki jih določajo robovi tega območja.

Najprej izločimo vrednosti gradienta na sliki, za katere velja, da je $Mag(G_x, G_y)$ blizu 0; primerna izbira je, da ven izločimo tiste gradiente, za katere velja $|G_x| < 0.3$ in $|G_y| < 0.3$. Izločene gradiente moramo sedaj nadomestiti z novimi vrednostmi, ki bodo dale zadovoljivo smer. To storimo tako, da interpoliramo okoliške vrednosti, ki so dobre. Ker pa dobre vrednosti ne ležijo nujno na mreži, ne moremo uporabiti kubične interpolacije na teh točkah. Zato potrebujemo interpolant, ki ne bo zahteval enakomerne razporeditve podatkov v x in y smereh. Uporabimo lahko interpolacijo s tankoploskovnim filtrom **??**.

Končno, za vsak center (c_x, c_y) poračunamo vrednosti gradienta s kubično interpolacijo. Kot poda katerim narišemo potezo, izračunamo z $\arctan(\frac{G_y}{G_x}) + 90\circ$. Slednje kote nato še perturbiramo. Uporaba normale na gradient nam da efekt, da so poteze čopiča prilepljene na objekt.

5.2.2 Slikanje z Beizerovimi krivuljami

Spreminjanje velikosti čopičev

Pogosto slikarji slikasjo tako, da najprej z debelejšim čopič naredijo grobo skico, nato pa z vedno manjšimi čopiči dodajo podrobnosti nas liko in delajo potrebne popravke. Čeprav ta postopek nima motivacije, ki bi bila vezana na računalniške algoritme, pa nam vrača estetsko primerne rezultate. Slikar tako lahko npr. uporablja drugčno tehniko, debelino čopiča ..., da naslika različne elemente na sliki (npr. zajčki na jasi). Uporaba potez iste oblike in barve na konstantnih območjih slike povzročijo to, da izgleda to območje umetno. V izogib temu slikar preko že narisanih potez, naslika manjše poteze, ki razgibajo sliko in poudarijo podrobnosti na sliki, ki jih slikar želi izpostaviti. Pri našemu postopku slikanja bomo z manjšim čopičem slikali le tam, kjer bo potrebno in ne povsod. Algoritem, ki ga bomo uporabljali je podoben piramidnemu algoritmu **??**, pri katerem pričnemo z grobo skico in nato dodajamo detajle z vedno manjšimi čopiči. Algoritem, ki ga bomo izpeljali, v resnici temelji na Laplaceovi piramidi: diferenčna slika L_i določa položaje potez čopiča.

Kakorkoli, diferenčna slika predvidi popolno rekonstrukcijo na nižjih ravneh piramide, naša rekonstrukcija pa je namerno nepopolna. Zato popravki na višjih ravneh povzročijo nezaželene artefakte. Algoritem, ki ga bomo predstavili, se temu problemu izogne tako, da diferenčno sliko izračuna na novo vsakič, ko uporabimo novo debelino čopiča.

Algoritem za vhodne podatke dobi sliko in seznam debelin čopiča. Debeline čopičev določimo z njihovimi polmeri R_i. Algoritem nato prične s slikanjem posameznih plasti, za vsak polmer čopiča svojo. Plasti barvamo od najdebelješega čopiča proti najtanjšemu. Osnovna plast je platno, pobarvano s konstantno barvo [2].

Za vsako plast P_i najprej ustvarimo referenčno sliko, ki jo dobimo tako, da zameglimo vhodno sliko (Gaussov filter s standardnim odklonom $f_\sigma R_i$), kjer je f_σ vnaprej določen parameter. Referenčna slika je sedaj slika, ki jo želimo čimbolje aproksimirati s čopičem trenutne velikosti. S tem, ko smo sliko zameglili, smo predpisali kako natančno želimo pobarvati sliko na tem nivoju. Ideja je v tem, da s posamezo velikostjo čopiča naslikamo podrobnosti, ki so v približni treutni velikosti čopiča. Za posamezno plast uporabimo vsakič isti algoritem, ki na podlagi trenutne diferenčne slike, doda tiste poteze čopiča, kjer aproksimacija za trenutno velikost ni zadovoljiva. Območja, ki se z referenčno sliko dovolj ujemajo (to nadzorujemo s parametrom T), ne spreminjamo več. S pomočjo parametra lahko dobimo bolj grobe slike (velik T) ali pa bolj fine (majhen T). V algoritmu 3 je povzeta psevdokoda ogrodja za algoritem.

Algoritem 3 Glavni algoritem.

Vhodni podatki: bla bla
Izhodni podatki: tralalal

```
1: function SLIKANJE(S, [R_1, ..., R_n])
2:     P ← pobarvamo z izbrano barvo
3:     for R_1, R_2 ... R_n do
4:         G ← S * G(f_σ R_i)
5:         POBARVAJPLAST(P, G, R_i)
6:     end for
7:     return P
8: end function
```

Prvo plast v točki 2. pobarvamo z barvo C; razlika med barvo C in katerokoli drugo barvo mora biti maksimalna. Vsako posamezno plast pobarvamo s preprosto zanko preko celotne slike. Postopek je podoben temu v **??**, pri katerem smo poteze čopiča razporedili enakomerno na mreži. Na tak način lahko zgrešimo robove in detajle med dvema zaporednima točkama na mreži. Pri našem algoritmu bomo zato naredili izboljšavo, ki bo namesto, da bi uporabila točko na mreži za center poteze, pregledala celotno okolico centra in poiskala točko v kateri je napaka največja. Vse poteze čopiča najprej določimo in jih šele nato, ko imamo vse določene renderiramo v izbranem vrstnem redu. Z naključno izbranim vrstnim redom se npr. lahko izognemo regularnosti. V algoritmu 4 je povzeta psevdokoda za ta algoritem.

V praksi ne bomo shranjevali seznama vseh potez, ki ga na kocu naključno premešamo ali pa mu določimo kateri drugi vrstni red, temveč lahko uporabimo t. i. Z-buffer.

[2] Platno ima lahko teksturo.

Algoritem 4 Pobarvaj plast.

Vhodni podatki: P, G, R

Izhodni podatki: Platno z na novo upodobljenimi potezami čopiča.

```
1: function PobarvajPlast(P, G, R)
2:     S ← nova množica potez
3:     D ← DiferenčnaSlika(P, G)
4:     g ← f_σ R
5:     x ← 0
6:     while x < širina slike do
7:         y ← 0
8:         while y < višina slike do
9:             M ← regija(x − g/2, x + g/2, y − g/2, y + g/2)
10:            napaka ← (∑_{i,j∈M} D_{i,j})/g^2
11:            y ← y + g
12:            if napaka > T then
13:                (x_m, y_m) ← argmax_{i,j∈M} D_{i,j}
14:                s ← Poteza(x_m, y_m, R, G)
15:                dodaj s množici potez S
16:            end if
17:        end while
18:        x ← x + g
19:    end while
20:    V izbranem vrstnem redu upodobi vse poteze s ∈ S na platno P.
21: end function
```

Ustvarjanje novih potez s čopičem

Individualne poteze čopiča na sliki imajo lahko svojo obliko, barvo, teksturo ... V nadaljevanju bomo predstavili način za generiranje dolgih ukrivljenih potez. Poteze čopiča bomo modelirali s pomočjo antializiranih B-kubičnih zlepkov, vsak s svojo barvo in debelino. Potezo čopiča upodobimo s krožno masko, ki jo vlečemo vzdolž B-zlepka.

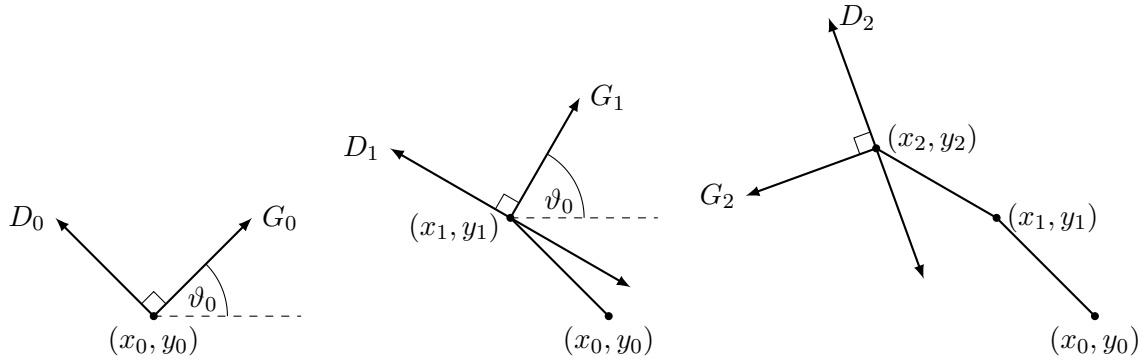
Poiskati moramo kontrolne točke za Bezierovo krivuljo. pričnemo v točki (x_0, y_0). Izbrani polmer čopiča je R. Potezo čopiča predstavimo s seznamom kontrolnih točk, barvo in debelino poteze.

Kontrolno točko (x_0, y_0) dodamo praznemu seznamu kontrolnih točk; barva poteze nastavimo na barvo slikovne točke (x_0, y_0) na referenčni sliki. Nato izračunamo naslednnjo točko vzdolž krivulje. Gradient Θ_0 v tej točki izračunamo iz Sobelovo filtrirane referenčne slike, ki smo jo predhodno iluminirali ($L(r, g, b) = 0.3{*}r{+}0.59{*}g{+}0.11{*}b$). Naslednjo točko (x_1, y_1) postavimo v smeri $\Theta_0 + \frac{\pi}{2}$ na razdalji R od (x_0, y_0). Poler čopiča R uporabljamo za razdaljo med kontrolnimi točkami, ker polmer R reprezentira količino podrobnosti, ki jo bomo zaobjeli s čopičem. Preostale kontrolne točke poračunamo po istem postopku. Proces iskanja novih kontrolnih točk prekinemo v naslednjih dveh primerih:

1. dosegli smo maksimalno število kontrolnih točk (l_{max});

2. barva kontrolne točke se od barve poteze razlikuje bolj kot od trenutne barve na platnu.

Z zaustavitvenim pogojem dolžine, se izognemu morebitni neskočni zanki. Za točko (x_i, y_i) izračunamo gradient v tej točki, Θ_i. Opazimo, da v tej točki, obstajajo dve normali na to točko, vsaka v svojo smer: $\theta_i + \frac{\pi}{2}$ in $\theta_i - \frac{\pi}{2}$. Izbereamo tisto smer normale, ki minimizira ukrivljenost poteze: izberemo smer D_i, za katero velja $\alpha(D_i, D_{i-1}) < \frac{\pi}{2}$.



(a) Poteza čopiča se začne v točki (x_0, y_0) in nadaljuje v smeri D_0, ki je normalna na G_0.

(b) V drugi točki (x_1, y_1) imamo dve normalni smeri na gradient: $\vartheta_1 - \frac{\pi}{2}$ in $\vartheta_1 + \frac{\pi}{2}$. Izbrali smo D_1, da zmanjšamo ukrivljenost krivulje.

(c) Ta postopek ponavljamo pri iskanju ostalih kontrolnih točk. Potezo čopiča bomo upodobili kot kubični B-zlepek s kontrolnimi točkami (x_i, y_i). Razdalja med kontrolnimi točkami je enaka trenutnemu polmeru čopiča.

Slika 5.1: Iskanje novih kontrolnih točk.

Ukrivljenost krivulje lahko nadziramo s infinite impulse response filter na smer poteze. Filter kontroliramo z vnaprej določeno filtrirno konstanto f_c. Za dano zadnjo smer $D'_{i-1} = (d'_{x,i-1}, d'_{y,i-1})$ in trenutno smer poteze $D_i = (d_x, d_y)$ izračunamo filtrirano smer kot

$$D'_i = f_c D_i + (1 - f_c) D'_{i-1} = (f_c d_{x,i} + (1 - f_c) d'_{x,i-1}, f_c d_{y,i} + (1 - f_c) d'_{y,i-1}).$$

V algoritmu 5 je podana psevdokoda za iskanje kontrolnih točk.

Minimalno dolžino za potezo l_{min} predpišemo, da se izognemo prekratkim potezam, ki bi delovale moteče. Za upodabljanje potez najprej s pomočjo subdivizij izračunamo zlepek. Nato pa vzdolž poti dobljenega zlepka narišemo potezo z antialiazirano ciklično masko.

5.3 Anizotropični čopič za likovno upodabljanje

Likovno upodabljanje s čopičem sestoji iz teh glavnih sklopov:

- postavitev potez na platno in njihovi parametri (velikost, prosojnost ...);
- izračun vektorskega polja, ki določi smeri potez in njihovo pot (množica urejenih točk);
- upodabljanje potez čopiča, ki pove kako bomo posamezno potezo upodobili na platnu.

Pogledali si bomo upodabljanje potez s čopičem.

Algoritem 5 Iskanje kontrolnih točk za kubični B-zlepek..

Vhodni podatki: x_0 y_0, R, G

Izhodni podatki: Seznam kontrolnih točk K.

function POTEZA(x_0, y_0, R, G)
  $C \leftarrow G.barva(x_0, y_0)$
  $K \leftarrow$ prazen seznam kontrolnih točk
  Dodamo točko (x_0, y_0) v seznam kontrolnih točk K.
  $(x, y) \leftarrow (x_0, y_0)$
  $(zd_x, zd_y) \leftarrow (0, 0)$
  for $i \in [1, \ldots, l_{max}]$ **do**
    if $i > l_{min}$ & $|G.barva(x, y) - P.barva(x, y)| < |G.barva(x, y) - C|$ **then**
      return K
    end if
    if $G.magnituda(x, y) == 0$ **then**
      return K
    end if
    $(g_x, g_y) \leftarrow G.smer(x, y)$
    $(d_x, d_y) \leftarrow (-g_y, g_x)$
    if $zd_x \cdot d_x + zd_y \cdot d_y < 0$ **then**
      $(d_x, d_y) = (-d_x, -d_y)$
    end if

    $(d_x, d_y) = f_c \cdot (d_x, d_y) + (1 - f_c) \cdot (zd_x, zd_y)$
    $(d_x, d_y) = (d_x, d_y)/\sqrt{d_x^2 + d_y^2}$
    $(x, y) = (x + R \cdot d_x, y + R \cdot d_y)$
    $(zd_x, zd_y) = (d_x, d_y)$
    Točko (x, y) dodamu seznamu K.
  end for
end function

5.3.1 Upodabljanje potez z anizotropičnim čopičem

Najpogostejša metoda za upodabljanje potez je risanje antialiazirane črte vzdolž poti poteze. Čeprav je to enostavno za implementacijo, pa se na ta način upodobljene poteze od ročno narisanih potez s čopičem močno razlikujejo. Antialiazirana črta ima le eno barvo, brez strukture. Predstavili bomo simulacijo ročnega risanja s čopičem. Najprej bomo predstavili model anizotropičnega čopiča. V maski čopiča bo vsaka vrednost večja od 0 predstavljala pripadajočo nit na čopiču. Inteziteta posameznih elemntov maske predstavljajo kontakt med niti čopiča in platnom (0 pomeni, da stika ni; 1 pomeni, da je popolni kontakt). Prosojnost čopiča je določena kot povprečna vrednost intezitet v maski čopiča. Vsaki niti čopiča najprej predpišemo barvo in nato vlečemo masko čopiča vzdolž poti poteze.

Digitalni čopič, ki smo ga opislai zgoraj nastavimo v treh korakih:

1. Izbrati moramo pravo velikost maske za čopič ali pa dano masko za čopiča skalirati, da ta ustreza velikosti čopiča.

2. Intezitete v maski čopiča pomnožimo s parametrom, ki smo ga nastavili, da prilagodimo inteziteto čopiča.

3. Posameznim nitim čopiča v maski čopiča priredimo barvo (barvo priredimo le tistim elemntom maske, ki imajo vrednost večjo od 0).

Barvo posameznim nitim čopiča določimo s formulo

$$C_{nit} = \alpha \cdot C_{nit} + (1 - \alpha) \cdot C_{poteza} + C_{šum},$$

kjer je C_{nit} barva na referenčni sliki, C_{poteza} je barva začetne kontrolne točke za potezo in $C_{šum}$ je perturbacija barve, s katero dosežemo učinek približnosti pri slikanju.

Po inicializaciji čopiča, narišemo potezo čopiča vzdolž poti, ki je sestavljena iz urejenih točk. Natančneje, na vsaki točki postavimo našo masko in pobarvamo območje, ki ga zavzema maska na sliki po formuli

$$C_{platno} = \frac{I_{nit}}{255} \cdot C_{nit} + (1 - \frac{I_{nit}}{255} \cdot C_{platno}),$$

kjer je C_{platno} matrika barv ustreznega območja na platnu in je I_{nit} maska čopiča z intezitetami

5.3.2 Lighting effect

Dodajanje učinka svetlosti in bleščanja platnu, lahko zelo izpopolni kvaliteto upodabljanja slik. Predstavili bomo metodo za dodajanje efekta svetljenja, ki je za implementacijo zelo enostavna. Podobno kot Hertzmann v **??**7], moramo najprej naračunati višinsko polje za platno, ki za vsako slikovno točko na platnu predstavlja njeno višino. Za razliko od Hertzmanna, ki vsaki posamezni potezi predpiše preslikavi, ki določata prosojnost in višino posameznih slikovnih točk na potezi, bomo uporabili masko čopiča in antialiazirano črto ter z upodabljanjem določili višinsko in prosojnostno preslikavo.

Inteziteto posameznih slikovnih točk na potezi čopiča (višinska preslikava), določimo kot povprečno vrednost intezitet, ki pri risanju s čopičem prečkajo to slikovno točko. Prosojnostno preslikavo pa določimo z upodobitvijo poteze čopiča na črno podlago kot antialiazirano črto, kateri nastavimo polmer nekoliko manj od polemra čopiča. Prosojnostne preslikave posameznih potez nato uporabimo kot uteži pri izračunu višinskega polja za platno.

Namesto uporabe Modela za senčenje Phong, bomo simulirali lighting effect tako, da bomo prilagodili intezitete slikovnih točk na platnu. Za vsako slikovno točko (x, y) na platnu, je prilagoditev intezitete te točke sorazmerna z višinsko razliko $D(x, y)$

$$\begin{aligned} d &= O(x, y) - \pi/2, \\ D(x, y) &= h(x + \cos d, y + \sin d) - h(x, y), \end{aligned}$$

kjer d predstavlja smer, ki je pravokotna na orientacijo poteze $O(x, y)$ in h označuje višino slikovne točke, tj. pripadajočo inteziteto slikovne točke na višinskem polju platna. Če je $D(x, y)$ pozitiven, potem bo vrednost intezitete slikovne točke narasla, sicer bo padla. Prilagoditev intezitet mora biti majhna vrednost, v primeru, ko je vrednost intezitete blizu 0 ali 255, po drugi strani je vrednost prilagoditve lahko ogromna, kadar je začetna

vrednost intezitete blizu povprečne vrednosti 127. Natančneje, za slikovno točko (x, y) na platnu, prilagoditev intezitet izračunamo kot

$$I(x,y) = I(x,y) + a \cdot D(x,y) \cdot \frac{\min(I(x,y), 255 - I(x,y))}{127.5},$$

kjer je a parameter, s katerim kontroliramo inteziteto lightinga. Ker je platno barvno, prilagoditev intezitete izvedemo za vsak barvni kanala posebaj.

5.3.3 Fast paint texture

Algoritem za vhod dobi urejen seznam potez čopiča, model senčenja in množico višinskih polj potez čopiča. Likovno upodabljanje tako poteka v treh korakih:

1. S kompoziranjem potez čopiča najprej izračunamo osnovno sliko.
2. Izračunamo višinsko polje, tj. za vsako slikovno točko na sliki pove skupno višino nanešene barve.
3. Končno sliko izračunamo s pomočjo bump-mappinga s pomočjo modela senčenja Phong.

Najprej moramo ustvariti matriko z vrednostmi barv, ki jih ima slika brez svetljenja. To naredimo s kompoziranjem potez čopiča na platno.

Sedaj izračunamo višinsko polje, ki pove višino nanešene barve za posamezne točke. Najprej nastavimo polje z ustreznimi dimenzijami in nastavimo barvo na črno. nato v vrstnem redu upodabljamo poteze čopiča, ampak kot črnobele. Toni slikovnih točk poteze so določeni s teksturno preslikavo. Vsakič celotni potezi dodamo zraven še vrednost, ki je sorazmerna s številom potez čopiča, ki so že narisane. Zato so poteze, ki so v ozadju narisane bolj temno (tj. plitva območja oz. dolince), poteze naisane kasneje pa bodo bolj svetlih barv. Če bi višinsko polje gradili tako, da bi kar prištevali zraven višine novo narisanih potez, bi se poteze narisane v ozadju pojavljale v višinskem polju, česar pane želimo.

Končno sliko izračunamo iz višinskega polja in osnovne slikae tako, da za izračunamo polje normal iz višinskega polje in poračunamo nove barve s pomočjo modela Phong.

Poglavje 6

Risanje z voščenkami

V članku [29] predstavljeni postopek likovnega upodabljanja risanja z voščenkami modelira risanje na podlagi fizikalnih modelov, ki so bili eksperimentalno določeni in preizkušeni. Delu voščenke s katerim rišemo po papirju pravimo *profil*. Voščenke so zgrajene iz mehkega in viskoznega materiala ter imajo standardni premer 8 mm (standardni večji premer meri 11 mm). Zaradi velikosti profila pri stiku voščenke z risalnim papirjem ne smemo privzeti, da se profil spreminja homogeno na celotni površini, temveč moramo obravnavati spreminjanje profila na mikroravni. Pri risanju z voščenkami je pomemben začetni profil voščenke, ki vpliva na kasnejše spreminjanje profila, ko se material na profilu zaradi hrapavosti papirja neenakomerno kruši v plasteh.

Najprej bomo v prvem razdelku določili podrobnejše modele za voščenke in risalni papir, ki jih bomo uporabili v algoritmu. V drugem razdelku bomo predstavili ogrodje glavnega algoritma za likovno upodabljanje z voščenkami ter definirali potrebne pojme in (fizikalne) količine, ki jih bomo potrebovali. Pri eksperimentalnih poskusih je bilo opaženo, da pri risanju z voščenko na delu papirja, kjer je že bila nanešena plast voska, star nanos voska v smeri risanja potisnemo v sosednje območje, nekaj pa se ga lahko prime tudi nazaj na profil ... Ta opažanja bomo fizikalno opisali in predstavili algoritme za njihovo simulacijo. V naslednjem razdelku bomo s pomočjo KM barvnega modela, predstavljenega v **??**, in eksperimentalno določenih parametrov za barve otroških voščenk predstavili končni algoritem, ki bo vrnil risbo narisano z voščenkami.

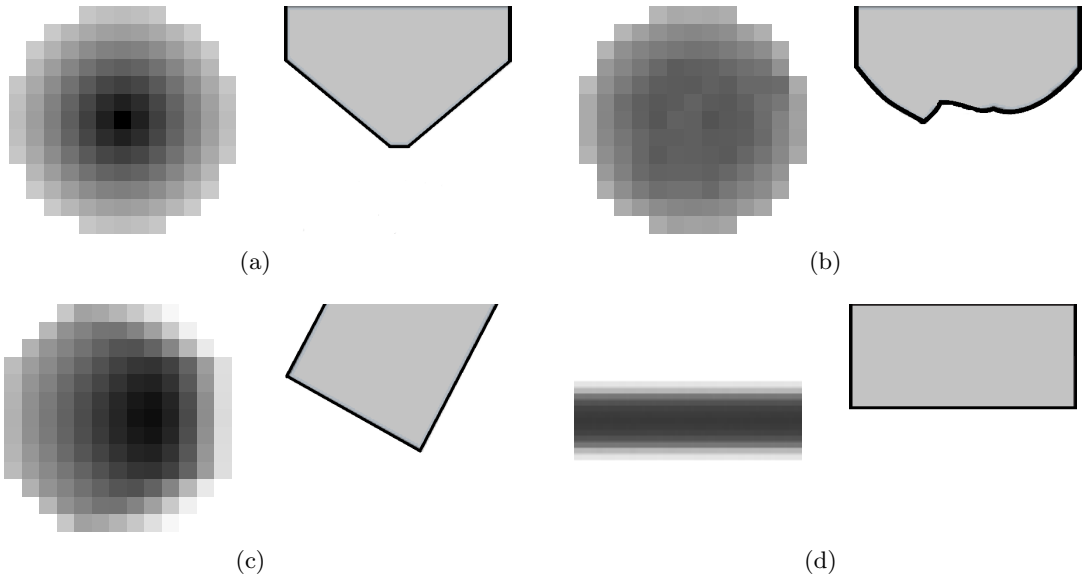
V zadnjem razdelku bomo opisali našo implementacijo algoritma in poskusno naredili slike z voščenkami, pri katerih uporabnik kot vhodni podatek ne bo podal skice v izbranih barvah za voščenke kot v članku, temveč bo podal (barvno) sliko, na podlagi katere bo program določil linije, ki jih bo pobarval z naračunanimi barvami voščenk.

6.1 Model voščenke in papirja

6.1.1 Model voščenke

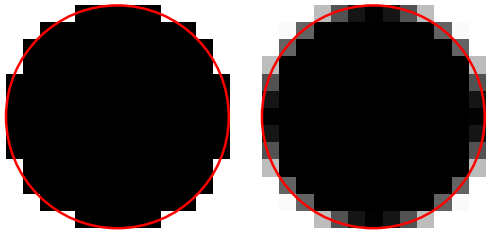
Profil voščenke modeliramo kot dvodimenzionalno višinsko masko M, pri čemer vrednost posamezne celice $m_{i,j} \in M$ predstavlja pravokotno oddaljenost $h_{m_{i,j}}$ pripadajoče točke na profilu voščenke od *osnovne ravnine* na kateri leži risalni papir. Maska M mora biti dinamična, saj se pri risanju profil spreminja zaradi luščenja voska. Spreminjanje maske bomo opisali pri obravnavi algoritma.

Uporaba dinamične maske nam omogoča izbiro poljubnega začetnega profila voščenke. Za profile voščenk, pri katerih kot pravokotno projekcijo na osnovno ravnino dobimo krog, pravimo, da so *okrogli profili*. Za slednje pravimo, da imajo *polmer* R, kadar je premer voščenke enak $2 \cdot R + 1$ (dodatna ena celica je tu iz tehničnih razlogov, saj pri implementaciji lažje operiramo z maskami lihe velikosti). Na sliki 6.2 so prikazani nekateri možni profili, ki jih lahko uporabimo pri modeliranju risanja z voščenkami.



Slika 6.1: Prikaz modelov za nekatere možne profile voščenke. Standardni profil voščenke je prikazan zgoraj levo. Intezitete slikovnih točk ustrezajo vrednostim $h_{m_{i,j}}$.

Celicam maske M predpišemo tudi statično vrednost $u_{m_{i,j}}$, ki je sorazmerna deležu površine celice $m_{i,j}$, ki leži znotraj območja pravokotne projekcije profila na osnovno ravnino. Vrednost $u_{m_{i,j}}$ poimenujemo *utež celice* $m_{i,j}$, za masko M pa pravimo, da je *utežena*. Z uporabo utežene maske se izognemu pojavu kvadratkastega videza končne slike (kar se pri dejanski izvedbi algoritma izkaže kot težava, še posebej pri uporabi profila voščenke na sliki **??**, kadar voščenko nagnemo pod kotom manjšim od 20°). Na sliki **??** je prikazan primer utežene maske okroglega profila voščenke s polmerom 6.

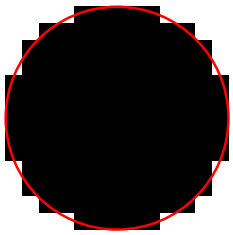


Slika 6.2: Prikaz modelov za nekatere možne profile voščenke. Standardni profil voščenke je prikazan zgoraj levo. Intezitete slikovnih točk ustrezajo vrednostim $h_{m_{i,j}}$.

6.1.2 Model papirja

6.2 Modeliranje risanja z voščenkami

Pri eksperimantalnih poskusih risanja z voščenkami opazimo, da se vosek s profila prenese na risalni list v neenakomernih plasteh in posledično spremeni obliko profila. Nadalje opazimo, da se vosek, ki je že nanešen na risalnem listu, če gremo ponovno preko njega z voščenko, prenese v sosednja lokalna območja v trenutni smeri risanja. Obravnava na mikroravni pokaže, da se prenese predvsem tisti del vosek, ki se nahaja na vrhih risalnega lista, in sicer v bližnje doline (glej siko **??**). Prav tako pa se lahko manjša količina voska prime nazaj na profil.



6.2.1 Osnovne količine in algoritem za risanje posameznih linij

Risanje posameznih linij simuliramo z algoritmom 6.

Algoritem 6 Povzetek glavnega algoritma za risanje z voščenkami.

Vhodni podatki: P_1, P_2, M, C, f, L
Izhodni podatki: Nova poteza z voščenko.

```
1: function NovaLinija(P_1, P_2, M, C, f, L)
2:     for P_i ∈ P_1P_2 do
3:         PrilagodiVoščenko(P, M, f, L)
4:         Redepozicija(P, V)
5:         Smearing(P_i, P_1P_2, M, L)
6:         DodajVosek(P_i, P_1P_2, f, M, C, L)
7:     end for
8: end function
```

Vhodni podatki algoritma so:

- P_1, P_2 ... začetna in končna točka linije, ki jo bomo narisali,
- M ... maska profila voščenke,
- C ... barva voščenke,
- f ... velikost sile s katero pritiskamo voščenko na risalni list pri risanju,
- L ... seznamov posameznih nanosov voska (višina nanosa in njegova barva) za vsako slikovno točko na papirju.

6.2.2 Prilagoditev profila voščenke

Na količino voska, ki se bo odluščil s profila, vplivajo naslednji dejavniki:

- površina profila,
- velikost in smer sile s katero pritiskamo voščenko pri risanju,
- relativna oddaljenost profila od risalnega lista,
- lokalna tekstura risalnega lista na trenutnem območju, kjer je profil voščenke.

Pri risanju z voščenko se s profila lušči vosek, kar pomeni, da se veča skupna oddaljenost profila od ravnine risalnega papirja. Ker ohranjamo silo f konstantno vzdolž ene linije, to pomeni, da bi na neki točki lahko izgubili stik med profilom in risalnim listom. Da se temu izognemu, bomo na vsakem koraku na novo prilagodili višino in obliko profila. Ob privzetku, da se dolžina voščenke zmanjšuje linearno, si bomo pri prilagajanju višine profila pomagali s Hookovim zakonom o kompresiji.

Hookov zakon

Hookov zakon lahko zapišemo v obliki

$$F = Y\frac{\Delta L}{L_0}A,$$

kjer so Y Youngova konstanta modula, ΔL konstanta za stiskanje, L_0 nestisnjena dolžina voščenke in A velikost profila. Ob privzetku, da je dolžina voščenke L_0 približno konstantna ($L_0 >> \Delta L$), lahko uvedemo novo konstanto λ, s katero se Hookov zakon prepiše v obliko:

$$F = \lambda A \Delta L. \tag{6.1}$$

Za konstanto λ izberemo vrednost, ki nam bo dala estetsko primerne rezultate.

Prilagoditev višine voščenke

Kot smo že omenili, je zaradi velikosti površine profila, potrebna obravnava profila na mikroravni. V ta namen bomo po formuli (6.1) izračunali za vsako posamezno celico maske $m_{i,j} \in M$ prispevek k celotni sili: $F_{m_{i,j}} = \lambda\delta h$ ($A = 1$ za vsako celico profila). V primeru, ko med celico profila in istoležno točko na risalnem papirju ni stika (celica profila leži nad istoležno točko na risalnem listu), prispevek k sili nastavimo na 0. Skupna sila F je enaka vsoti posameznih prispevkov, $F = \sum_{m_{i,j}} F_{m_{i,j}}$.

Količina δh je v tem primeru razlika med višinama celice profila in istoležne točke na risalnem papirju skupaj z že nanešenim voskom. Višina celice profila bo tu odvisna od višine voščenke h, posledično je torej F funkcija spremenljivke h, $F(h)$. Dobili smo aproksimacijski problem, pri katerem želimo čimbolje aproksimirati velikost sile f. Ker je F vsota posameznih prispevkov, funkcija $F(h)$ ni več zvezna, lahko pa jo opišemo kot deloma zvezno funkcijo. Pri aproksimaciji si bomo pomagali z Newtonovo metodo. Natančnost aproksimacije in same simulacije pa nadziramo s konstanto napake ε. Eksperimentalno opazimo, da uporaba večje napake pohitri Newtonovo metodo, še vedno pa nam da likovno zadovoljive rezultate. V algoritmu 7 je podana psevdokoda za določitev nove višine voščenke.

Algoritem 7 Prilagoditev voščenke.

Vhodni podatki: P, M, f, L

Izhodni podatki: Spremenjena maska voščenke.

function PRILAGODITEVVOŠČENKE(P, M, f, L)
  $h_{min}^{voščenka} \leftarrow \min_{m_{ij} \in M} h_{m_{ij}}$
  $h_{min} \leftarrow \min_{m_{ij} \in M} h_{P_{ij}}$ ▷ $P_{ij} = P + (i, j)$
  $h_{max} \leftarrow \max_{m_{ij} \in M} h_{P_{ij}}$
  while $h_{max} - h_{min} > \Delta$ **do**
    $h_{mid} \leftarrow (h_{max} - h_{min})/2$
    $f_{mid} \leftarrow 0$
    for all $m_{ij} \in M$ **do**
      $\delta h = h_{P_{ij}} + h_{L_{P_{ij}}} - (h_{m_{ij}} - h_{min}^{voščenka} + h_{mid})$
      if $\delta h > 0$ **then**
        $f_{mid} \leftarrow \lambda \delta h$
      end if
      if $f < f_{mid}$ **then**
        $h_{min} \leftarrow h_{mid}$
      else
        $h_{max} \leftarrow h_{mid}$
      end if
    end for
    $h_{mid} \leftarrow (h_{min} + h_{max})/2$
    for all $h_{m_{ij}}$ **do**
      $h_{m_{ij}} \leftarrow h_{m_{ij}} - h_{min}^{voščenka} + h_{mid}$
    end for
  end while
end function

6.2.3 Trenje

Pri risanju z voščenko se bo zaradi mehkobe materiala iz katerega je zgrajena voščenka in trenja, ki nastane med voščenko in papirjem, vosek razporedil naokoli. Trenje med voščenko in papirjem bomo opisali na makro in mikro ravni. Na makro ravni nas zanima normalna sila voščenke na površino papirja. Če voščenka preide mimo konveksnega delčka v teksturi papirja, se bo od voščenke odstranil del voska, ki bo ostal za tem konveksnim delčkom papirja. Na mikro ravni uporabimo koeficient trenja, da aproksimiramo hrapavost papirja na manjši skali. Želimo, da je količina prestrukturiranega voska sorazmerna s silo trenja, ki je definirana kot

$$\vec{F_F} = \mu \vec{F_N} = \mu \vec{N} \frac{\vec{N} \cdot \vec{F_C}}{\left\|\vec{N}\right\| \left\|\vec{F_C}\right\|},$$

kjer je $\vec{F_C}$ sila voščenke na površino papirja, $\vec{F_F}$ je sila trenja, $\vec{F_N}$ je normala sile trenja na površino papirja, $\vec{N}$ je normala na površino papirja in μ je koeficient trenja papirja pri stiku z voščenko.

Na podlagi teksture papirja interpoliramo sosednje vrednosti višin celic papirja, da dobimo ravnino po kateri se voščenka premika, in izračunamo silo trenja. Sila trenja je odvisna od naračunane ravnine, dane konstante sile trenja voščenke na papir in smeri risanja z voščenko.

Vrednost μ je odvisna od tega ali rišemo na čist papir ali pa je ta že porisan. Še več, območje papirja na katerem je debelina nanešenega voska voščenke večja, bo imelo drugačne lastnosti in trenje kot območja papirja s tanjšim slojem nanešenega voska voščenke. Če bi želeli biti rigorozni pri simulaciji trenja med voščenke in papirja, bi morali upoštevati zgornje. Vendar pa zaenkrat nimamo še rezultatov, ki bi povedali vrednost koeficienta za trenje papirja, ki bi dal rezultate kot bi jih želeli. Na področju teg bla bla **??** Pri naši simulaciji bomo določili posebej koeficient trenja za voščenko $\mu_{voščenka}$ in papir μ_{papir}. Koeficient trenja bomo izračunali sproti kot linearno kombinacijo $\mu_{voščenka}$ in μ_{papir} na podlagi trenutnega faktorja.

V algoritmu 8 je podana psevdokoda za risanje z voščenko z upoštevanjem trenja.

Algoritem 8 Risanje z voščenko in računanje trenja med voščenko in papirjem.

Vhodni podatki: P, $\vec{V}$, M, C, f, L
Izhodni podatki: Dodan vosek.

function DodajVosek(P, $\vec{V}$, M, C, f, L)
  $\vec{V} \leftarrow \vec{V} / \max(x_{\vec{V}}, y_{\vec{V}})$
  for all $m_{ij} \in M$ **do**
    $P_{ij} \leftarrow P + (i, j)$
    $\hat{P}_{ij} \leftarrow P_{ij} + \vec{V}$
    $\vec{S}_{ij} \leftarrow (x_{\vec{V}}, y_{\vec{V}}, -h_{P_{ij}})$
    $\vec{F}_{ij} \leftarrow (x_{\vec{V}}, y_{\vec{V}}, -f)$
    $\vec{F}_{ij} \leftarrow 1/(1 + h^{voščenka}_{\hat{P}_{ij}})$
    $\mu_{ij} \leftarrow \alpha\mu_{papir} + (1 - \alpha)\mu_{voščenka}$
    $\delta h^{voščenka}_{\hat{P}_{ij}}) \leftarrow \mu(h_{\hat{P}_{ij}} - h_{m_{ij}}) \sin(\vec{S}_{ij}, \vec{F}_{ij})$
    $h_{m_{ij}} \leftarrow h_{m_{ij}} + \delta h^{voščenka}_{\hat{P}_{ij}})$
    $L_{P_{ij}} \leftarrow L_{P_{ij}} + \left\{(\delta h^{voščenka}_{\hat{P}_{ij}}), C)\right\}$
  end for
end function

6.3 Smearing

Smearing je lastnost voščenke, podobno kot je krvavanje pri čopiču. Pri risanju z voščenko na območjih z že nanešenim voskom voščenke, se vosek porazdeli in potisne bolj noter v luknje. Za simulacijo smerariga bomo najprej uvedli smearing masko, ki upošteva trenutno točko in njene sosede. Vrednosti v maski voščenke določajo količino voska, ki se bo prenesel izpod trenutne točke in raznesel naokoli. Pri tem torej upoštevamo točko in njenih osem sosedov. Zaradi mehkobe materiala voščenke, predpostavimo,da je to dovolj upošteavti, medtem ko za nekatere druge materiale to ne bi bilo res (npr. pastelne barvice).

Pri simulaciji si bomo pomagali s filtrom S velikosti 3×3. Elemente filtra izračunamo

s formulo

$$S_{xy} = \frac{1}{\|(x,\vec{y})\|}(\alpha\Delta z + \beta(\widehat{x,y}) \cdot \hat{V}).$$

Vrednost celice S_{xy} nastavimo na 0, preprečimo morebitni prenos voska nazaj na voščenko.

V algoritmu 9 je podana psevdokoda za postopek smearinga.

Algoritem 9 Smearing.

Vhodni podatki: P, $\vec{V}$, M, L
Izhodni podatki: Posmearan papir.

function SMEARING(P, $\vec{V}$, M, L)
  $\vec{V} \leftarrow \vec{V}/\max(x_{\vec{V}}, y_{\vec{V}})$
  for all $m_{ij} \in M$ **do**
    $P_{ij} \leftarrow P + (i, j)$
    $\hat{P}_{ij} \leftarrow P_{ij} + \vec{V}$
    $\vec{S}_{ij} \leftarrow (x_{\vec{V}}, y_{\vec{V}}, -h_{P_{ij}})$
    $\vec{F}_{ij} \leftarrow (x_{\vec{V}}, y_{\vec{V}}, -f)$
    $\vec{F}_{ij} \leftarrow 1/(1 + h^{voščenka}_{\hat{P}_{ij}})$
    $\mu_{ij} \leftarrow \alpha\mu_{papir} + (1 - \alpha)\mu_{voščenka}$
    $\delta h^{voščenka}_{\hat{P}_{ij}}) \leftarrow \mu(h_{\hat{P}_{ij}} - h_{m_{ij}}) \sin(\vec{S}_{ij}, \vec{F}_{ij})$
    $h_{m_{ij}} \leftarrow h_{m_{ij}} + \delta h^{voščenka}_{\hat{P}_{ij}})$
    $L_{P_{ij}} \leftarrow L_{P_{ij}} + \left\{(\delta h^{voščenka}_{\hat{P}_{ij}}), C)\right\}$
  end for
end function

6.4 Redepozicija

Ena izmed lastnosti voščenk je tudi ta, da se vosek, ko prečkamo območje, na katerem smo predhosno že nanesli plast voska, vosek nanese nazaj na voščenko. Pri premikanju voščenke po papirju se vosek, ki je že nanešen odkruši s papirja in lahko gre nazaj na voščenko ter se nato z voščenko prenese na drugo območje na papirju. Podoben problem je pri risanju s čopičem **??**.

Zaradi viskoznosti materiala voščenke, se pigmenti v voščenki ne mešajo tako kot pri barvi, temveč se plasti voščenke odluščijo s papirja in pritisnjeo nazaj na voščenko. Ta vosek bo bil razporejen naokoli linearno in ne eksponentno. Barva s čopiča se prične mešati z barvo na platnu takoj ob stiku čopiča s platnom. Medtem, ko se pri voščenki nič voska ne bo razporedilo zgolj samo pri stiku.Z voščenko moramo podrgniti vosek iz papirja.

V algoritmu 10 je povzeta psevdokoda za redepozicijo voska.

6.5 Upodabljanje z voščenkami

Vosek je najlažje obravnavati kot prosojen pigment, za kar sta enostavna modela RGB in CMY pomankliva, da bi ju lahko uporabili. Namesto tega bomo uporabili Kubelka-Monk barvni model **??**. KM model aproksimira spektralno prosojnost, sipanje in motnje.

Algoritem 10 Redepozicija voska.

Vhodni podatki: P, $\vec{V}$, M, L, f
Izhodni podatki: Razporejen vosek.

function REDEPOZICIJA(P, $\vec{V}$, M, L, f)
  for all $m_{ij} \in M$ **do**
    $S \leftarrow 3 \times 3$ smearing mask
    for all $s_{qr} \in S$ **do**
      s_{qr} $\max\left\{0, \vec{V} \cdot \hat{(q,r)}\right\}$
    end for
    $S \leftarrow \gamma f S / (\sum s_{qr})$
    $\delta h_{voščenka} \leftarrow \max\left\{0, h_{m_{i+q,j+r}} - h_{m_{ij}}\right\}$
    $L'_{P_{ij}} \leftarrow \{l_a, \ldots, l_n\} : \sum h_{l_i} \leq \delta h_{voščenka}$
    $L_{P_{ij}} \leftarrow L_{P_{ij}} - L'_{P_{ij}}$
    for all $s_{qr} \in S$ **do**
      for all $l_k \in \delta L'_{P_{ij}}$ **do**
        $m_{i+q,j+r} \leftarrow m_{i+q,j+r} + s_{qr} l_k$
      end for
    end for
  end for
end function

Vrednosti teh lastnosti lahko določimo s pomočjo dveh barv **??**. Vsaka izmed teh barv je opaženi rezultat nanešene plasti na enobarvno ozadje (uporabimo belo in črno ozadje). Iz teh dveh barv KM model nato interpolira ti dve vrednosti, da dobi vrednosti za poljubne debeline nanešenega voska na ozadje katerekoli barve. KM model naredi to tako, da predvidi količino sipanja svetlobe glede na material pigmenta, in koliko je ta material prosojen. KM model aproksimira tudi spremembe, ki nastanjeo zaradi tankih motenj.

V našem modelu, ki ga bomo uporabili za risanje z voščenkami, bomo zanemarili motnje, saj v eksperimntalnih poskusih ni zaznati bistvenega vpliva. Posledično je vsaka voščenka predstavljena z enim naborom RGB barv ter parametrom prosojnost in sipanje.

Kot smo že omenili prej zelo tanke plasti voščenke združimo skupaj. Optične lastnosti končne plasti nastavimo na uteženo povprečje združenih plasti. Vsak prispevek plasti h končni plasti je torej sorazmeren z njegovo višino. To je groba peonostavitev KM modela, vendar še vedno da sprejemljive rezultate in bistveno zmanjša čas, ki je potreben za smearing.

Za upodobitev slike z voščenkami moramo vsaki točki na papirju P_{ij} pripisati barvo $C_{P_{ij}}$. Za izračun barve, ki jo dobimo za plast $l_k \in L_{P_{ij}}$, uporabimo barvo plasti C_{l_k}, njeno prosojnost t_{l_k} in sipanje r_{l_k}.

V algoritmu 11 je podana psevdokoda za upodabljanje slik z voščenkami.

Eksperimentalno smo določili osnovne barve vočenk in njihove parametre. Podani so v tabeli 6.1.

Algoritem 11 Algoritem za upodabljanje slik z voščenkami.

Vhodni podatki: T

Izhodni podatki: Slika narisana z voščenkami.

1: **function** UPODABLJANJE(T)
2: **for all** P_{ij} teksture papirja T **do**
3: $C_{ij} \leftarrow C_{P_{ij}}$
4: **for all** plasti voska l_k v točki P_{ij} **do**
5: $C_{ij}^{prosojnost} \leftarrow (t_{l_k} C_{l_k})^{h_{l_k}} C_{ij}$
6: $C_{ij}^{sipanje} \leftarrow 1 - (1 - C_{l_k})^{r_{l_k} h_{l_k}}$
7: $C_{ij} \leftarrow C_{ij}^{prosojnost} + C_{ij}^{sipanje}$
8: **end for**
9: **end for**
10: **end function**

Voščenka	Barva	R	G	B	t	s
	rdeča	0.95	0.45	0.45	0.605	0.0425
	oranžna	0.999	0.55	0.3	0.77	0.03
	rumena	0.95	0.9	0.2	0.869	0.0425
	zelena	0.35	0.8	0.35	0.55	0.06
	modra	0.3	0.5	0.9	0.77	0.045
	vijolična	0.65	0.45	0.75	0.715	0.05
	rjava	0.8	0.6	0.55	0.495	0.075
	črna	0.26	0.25	0.245	0.935	0.05
	siva	0.42	0.4	0.39	0.594	0.275
	bela	0.8	0.8	0.78	0.88	0.175
	periwinkle	0.7	0.7	0.9	0.605	0.125
	morsko zelena	0.6	0.9	0.65	0.55	0.1
	orhideja	0.85	0.4	0.84	0.88	0.075

Tabela 6.1: Tabela z vrednostmi za barve, ki jih uporabljajo otroci (povzeto po članku **??**).

Poglavje 7

Risanje s svičnikom

V zadnjem poglavju bomo predstavili algoritem za risanje s svinčnikom. Osnova naše implementacije algoritma bo temeljila na članku [], ki ga bomo prilagodili in dopolnili z nekaterimi dodatnimi koraki iz članka [].

Kot smo videli že v razdelku **??**, lahko pristopimo k risanju s svinčnikom na več načinov. V članku [] predstavljeni postopek risanja sodi v kategorijo risarskih postopkov, kjer senčenje izvedemo tako, da ustvarimo zvezno območje sivine, ki vsebuje teksturo, z linijskimi črtami pa poudarimo linije, ki dajo objektom na sliki obliko. Predstavljeni postopek risanja s svinčnikom v tem članku bomo razdelili v tri faze:

1. zaznava linij na sliki in izračun linijske slike $\tilde{S}$;
2. izračun sivinske slike T;
3. kompozicija linijske in sivinske slike v skupno risbo R.

Posamezne faze bomo predstavili v prvih treh razdelkih, v zadnjem razdelku pa bomo opisali našo implementacijo tega algoritma in njegovo analizo skupaj z dodatnimi primeri.

7.1 Linijska slika

Pri risanju s svinčnikom risarji uporabljajo kratke linijske črte. Dolge linijske črte na risbi narišejo kot zaporedje krajših, ki se na koncih nekoliko križajo. Podoben pristop bomo izbrali v našem algoritmu za izračun linijske slike. Končni rezultat, ki ga bomo dobili, bo neosenčena skica vhodne slike.

Za izračun linijske slike bomo morali na vhodni sliki zaznati linije, ki jih želimo narisati, določiti njihov položaj, dolžino, odtenek sivine in debelino. Zaradi prisotnosti teksture in šuma na vhodni sliki, običajni filtri za zaznavo robov na sliki, npr. Sobelov filter, ne bodo zadostili likovnim kriterijem. Zato bomo pripravili lastne filtre, ki bodo dali zadovoljiv rezultat tudi v primeru prisotnosti teksture in šuma. V nadaljevanju bomo opisali posamezne korake v algoritmu za risanje linij in jih opremili s primeri.

7.1.1 Classification

Vhodno (barvno) sliko moramo najprej pretvoriti v sivinsko sliko I. Uporabimo lahko standardno sivinsko pretvorbo ali pa naredimo svetlostno pretvorbo. Eksperimentalno se

izkaže, da uporaba svetlostne pretvorbe ponuja boljše izhodišče za zaznavo robov na sliki, zato bomo v našem algoritmu izbrali to pretvorbo. Na dobljeni sliki I bomo nato uporabili še filter za izostritev robov, ki bo linije na sliki naredil bolj izrazite.

Podobno kot v razdelku **??**, ko smo izpeljevali filter za zaznavo robov, bomo tudi sedaj s pomočjo gradientnih slik v x in y smereh ($\partial_x I$ in $\partial_y I$) izračunali gradientno sliko

$$G \coloneqq \sqrt{(\partial_x I)^2 + (\partial_y I)^2}\ .$$

Kot smo videli pri filtrih za zaznavo robov, znajo ti zaznati robove v treh smereh (vodoravno, navpično, diagonalno), kar pa z likovnega stališča ni zadostno. Pri risanju namreč linijske črte rišemo v poljubnih smereh. Zato bomo definirali množico pomožnih filtrov $\mathfrak{L}_n = \{\mathcal{L}_i\}_i^n$, ki bodo znali zaznati robove v n smereh (posledično bomo znali narisati linijske črte v teh n smereh). Posamezen filter predstavlja linijsko črto v i-ti smeri z dolžino, ki je sorazmerna z velikostjo vhodne slike. V našem algoritmu bomo uporabili 4 oz. 8 različnih smeri, imeli bomo torej $\mathfrak{L}_4 = \{\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3, \mathcal{L}_4\}$ oz. $\mathcal{L}_8 = \{\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_8\}$. Velikost filtrirnih matrik bomo nastavili na $\frac{1}{30}$ velikosti slike.

Zgornje filtrirne matrike bomo sedaj uporabili za izračun množice *odzivnih slik* $\{G_i\}_i^n$, ki ima elemente G_i definirane s formulo

$$G_i \coloneqq \mathcal{L}_i \star G\ .$$

Odzivna slika G_i je torej matrika velikosti vhodne slike, ki ima shranjene podatke o količini. Za vsak piksel torej dobimo na ta način podatke o 8 oz. 16 smereh. Izmed teh vrednosti bomo sedaj izbrali tisto vrednost, ki je največja:

$$C_i(p) = \begin{cases} G(p) & \text{če } \operatorname{argmax}_i \{G_i(p)\} = i, \\ 0 & \text{sicer.} \end{cases}$$

Z argmax smo si zagotovili, da bomo res imeli eno samo smer (i guess). Z vsako smer na ta način določimo magnitudo slike v tej smeri. Med G in množico $\{C_i\}_i$ obstaja zveza $\sum_i C_i = G$. Ker smo za posamezen pregledali njegovo okolico in poračunali gosto množico smeri, smo dobili smeri za piksel, ki so na šum na sliki (pri tem ni mišljen samo šum, npr. beli šum, temveč tudi strukture, teksture na originalni sliki, ...).

7.1.2 Line shaping

Sedaj, ko imamo za dane smeri na sliki mape gradientov, bomo za vsak piksel generirali linije oz. črtice s pomočjo konvolucije:

$$S' = \sum_i (\mathfrak{L}_i \star C_i).$$

Vrednosti v dobljenem S' sedaj invertiramo in jih slikamo v interval $[0, 1]$. Pokažemo primere, ko se sekajo linije. Pokažemo primer za množico C_i. Pogledamo razliko, če uporabimo luminance ali navadno. Pogledamo razliko, če izostrimo in če ne izostraimo. Pogledamo primer, ko uporabimo različne velikosti za naše konvolucijske matrike itd. Na ta način smo dobili skico slike oz. linije. Risarji pa poleg risanja linij slike, uporabljajo še senčenje. Senčenje lahko izvedemo tako, da uporabimo hatching. V članku je predstavljen nov način dodajanja sence. Namreč uporabimo kar ton slike.

7.2 Tone drawing

7.2.1 Tone Map Generation

Spet bomo prvotno sliko najprej s pomočjo filtra pretvorili v črnobelo sliko. Za vsak piksel tako dobimo informacijo o njegovi inteziteti oz. stopnji sivine. Vrednosti pikslov preslikamo na interval $[0, 1]$, da se bomo izognili problemom pri računanju distribucije oz. verjetnosti. Predstavljen je model senčenja s pomočjo tonske distribucije risbe. Za razliko od visoko variabilnih tonov na originalni sliki, risba ponavadi zadošča nekemu modelu.

Primer 7.1 Primer fotografije in pa risbe. Ter pripadajoča histograma.

To se zgodi zato, ker risba nastane kot posledica stika med papirjem in svinčnikom (grafitom), ki večinoma sestoji iz dveh glavnih tonov. Za zelo svetla območja na originalni sliki, risar tako ne nariše ničesar, Za linije in poudarjena območja risar poudari z grafitom ta območja. Za ostala območja pa uporabi srednje močne tone sivine, da ustvari teksturo in bogatost risbe.

7.2.2 Model-based Tone Transfer

Predstavili bomo parametrični model, ki predstavlja končno tonsko distribucijo, zapisano kot

$$p(v) = \frac{1}{Z} \sum_{i=1}^{3} \omega_i p_i(v),$$

kjer je v vrednost sivine in je $p(v)$ verjetnost, da je piksel na sliki pobarvan s tonom v. Z je normalizacijska vrednost, ki poskrbi, da je vrednost integrala $\int_0^1 p(v)dv = 1$. Sivinsko lestvico razdelimo na tri dele glede na vrednosti sivine. Verjetnosti p_i pripadajo posameznim delom. Vrednosti ω_i pa so določena tako, da potem, ko razdelimo tonsko distribucijo v tri dele, preštejmo število pikslov, ki padejo v posamezni del tonske lestvice.

Primer 7.2 Na sliki so prikazani je za dano risbo prikazana distribucija. Vsi trije layerji. Ter distribucije za posamezne layerje.

Kot smo videli v zgornjem primeru je glavna razlika med naravnimi slikami in pa risbami ta, da slednja sestoji iz bolj svetlih območij, ki je barva papirja.

Da bi modelirali distribucijo p_1, ki pripada svetlejšemu delu tonske lestvice, si bomo pomagali z Laplaceovo porazdelitvijo. Slednja bo imela vrh pri vrednosti 255 (vrednost določimo s pomočjo eksperimenta), kajti piksli se skoncetrirajo okoli te vrednosti in v tej vrednosti vrednost porazdelitve p ostro naraste. Nekaj variacije nastane zaradi uporabe radirke. Definiramo

$$p_1(v) = \begin{cases} \frac{1}{\sigma_b} e^{-\frac{1-v}{\sigma_b}} & \text{če } v \leq 1, \\ 0 & \text{sicer.} \end{cases}$$

Pri tem je vrednost σ_b parameter, ki določa faktor (scale) distribucije.

Za razliko od svetlejšega dela tonske lestvice, srednjetonska lestvica nima (nujno) vrha pri nobeni vrednosti. Zato bomo to porazdelitev zapisali s pomočjo normalne porazdelitve:

$$p_2(v) = \begin{cases} \frac{1}{u_b - u_a} & \text{če} u_a \leq v \leq u_b, \\ 0 & \text{sicer.} \end{cases}$$

Pri tem sta u_a in u_b parametra, ki vplivata na širino porazdelitve.

Temnejši del tonske lestvice pa bomo spet modelirali s pomočjo Laplaceove porazdelitve:

$$p_3(v) = \frac{1}{\sqrt{2\pi\sigma_d}} e^{-\frac{(v-\mu_d)^2}{2\sigma_d^2}}.$$

Pri tem je μ_d srednja vrednost in σ_d je razpršenost. Variacija pri temnejšem delu tonske lestvice je običajno širša kot pa tista pri svetlejšem delu lestvice.

7.2.3 Parameter learning

Parametri, ki nastopajo v formulah za porazdelitve p_i, kontrolirajo obliko histogramov za posamezne dele tonske lestvice.

Najprej vhodno sliko pretvorimo v črnobelo, da dobimo I. Nato sliko I rahlo zameglimo s pomočjo Gaussovega filtra. V nadaljevanju moramo za slikovne točke na I določiti v katerega od treh delov sodi posamezen piksel, glede na njegovo vrednost. Za temni in svetli del imamo vnaprej določene tresholde, preostali piksli pa padejo v srednji del sivinske lestvice. Število pikslov, ki padejo v posamezne dele tonske lestvice, določajo uteži ω. Za vsak del tonske lestvice določimo srednjo vrednost m in pa standardno deviacijo s, vrednosti parametrov določimo z uporabo Maximum Likelihood Estimation (MLE) ter dobimo parametre zapisane v obliki zaprtih formul:

$$\begin{aligned}
&\sigma_b = \frac{1}{N}\sum_{i=1}^{N} |x_i - 1|\,; \\
&u_a = m_m - \sqrt{3}s_m,\ u_b = m_m + \sqrt{3}s_m; \\
&\mu_d = m_d,\ \sigma_d = s_d.
\end{aligned}$$

Pri tem je N število vseh pikslov, x_i pa so vrednosti posameznih pikslov.

7.2.4 Pencil Texture Rendering

Generiranje ustrezne strukture s svinčnikom je zahteven problem. Pri rendiranju teksture bomo uporabili naračanunano tonsko sliko in uporabili teksturo, ki se je naučimo iz dejanskih risb. zato bomo potrebovali zbirko tonskih slik. Za vsako risbo bomo sicer potrebovali le eno samo. Pri risanju človek doseže senčenje tako, da riše s svinčnikom na istem mestu. Na ta način sliko potemni. Ta proces lahko simuliramo tako, da uporabimo eksponentno obliko $H(x)^{\beta x} \approx J(x)$ oz. v logaritemski domeni to pride $\beta(x)\ln H(x) \approx \ln J(x)$. Interpretacija tega je, da teksturo H na istem mestu narišemo β-krat, da dosežemo lokalno vrednost tona v J. Veliki β tako pomeni, da bomo sliko bolj potemnili, medtem ko majhen β pomeni, da bomo sliko ohranili bolj svetlo. Zahtevamo tudi, da je β lokalno gladka. β je rešitev enačbe

$$\beta^* = \mathrm{argmin}_\beta \left\|\beta \ln H - \ln J\right\|_2^2 + \lambda \left\|\Delta\beta\right\|_2^2,$$

kjer je λ utež z vrednostjo 0.2 v našem poksusu (njihovem). Zgornjo enačbo lahko pretvorimo v linearno obliko, ki jo lahko rešimo s pomočjo konjugiranih gradientov (conjugate gradient).

Končno teksturo T izračunamo s pomočjo eksponentne funkcije kot $T = H^{\beta^*}$.

Končno risbo dobimo tako, da pomnožimo to teksturo z linijami iz prejšnjega poglavja: $R = S \cdot T$. Pri tem gre za množenje po komponentah.

7.3 Color Pencil Drawing

Pretvorimo v YUV barvni prostor in uporabimo R namesto kanala Y. Potem pa zopet pretvorimo nazaj v RGB barvni prostor.

Literatura

[1] K. Perlin A. Hertzmann, *Painterly Rendering for Video and Interaction*, NPAR '00 Proceedings of the 1st international symposium on Non-photorealistic animation and rendering, ACM New York, NY, USA, 2000, str. 7–12.

[2] A. Appel, F. J. Rohlf, A. J. Stein, *The haloed line effect for hidden line elimination*, SIGGRAPH '79 Proceedings of the 6th annual conference on Computer graphics and interactive techniques, ACM New York, NY, USA, 1979, str. 151–157.

[3] A. Buades, B. Coll, J. Morel, *A Non–Local Algorithm for Image Denoising*, CVPR '05 Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, IEEE Computer Society Washington, DC, USA, 2005, str. 60–65.

[4] A. Buades, B. Coll, J. M. Morel, *Neighborhood Filters and PDE's*, Numerische Mathematik **105** (2006), 1–34.

[5] H. Cohen, *The Further Exploits of Aaron, Painter*, Stanford Humanities Review **4** (1995), 141–158.

[6] C. J. Curtis, S. E. Anderson, J. E. Seims, K. W. Fleischer, D. H. Salesin, *Computer-generated Watercolor*, SIGGRAPH '97 Proceedings of the 24th annual conference on Computer graphics and interactive techniques, ACM Press/Addison-Wesley Publishing Co. New York, NY, USA, 1997, str. 421–430.

[7] E. Daniels, *Deep canvas in Disney's Tarzan*, SIGGRAPH '99 ACM SIGGRAPH 99 Conference abstracts and applications, ACM New York, NY, USA, 1999, str. 200–200.

[8] M. W. Frazier, *An Introduction to Wavelets Through Linear Algebra*, Springer–Verlag New York, New York, USA, 1999.

[9] P. Haeberli, *Paint By Numbers: Abstract Image Representations*, SIGGRAPH '90 Proceedings of the 17th annual conference on Computer graphics and interactive techniques, ACM New York, NY, USA, 1990, str. 207–214.

[10] J. Hays, I. Essa, *Image and Video Based Painterly Animation*, NPAR '04 Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering, ACM New York, NY, USA, 2004, str. 113–120.

[11] A. Hertzmann, *Painterly Rendering with Curved Brush Strokes of Multiple Sizes*, SIGGRAPH '98 Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, ACM New York, NY, USA, 1998, str. 453–460.

[12] A. Hertzmann, *Paint By Relaxation*, Tech. report, New York University, NY, USA, 2000.

[13] A. Hertzmann, *Fast Paint Texture*, NPAR '02 Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering, ACM New York, NY, USA, 2002, str. 91–ff.

[14] A. Hertzmann, *Non-Photorealistic Rendering and the Science of Art*, NPAR '10 Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering, ACM New York, NY, USA, 2012, str. 147–157.

[15] H. Huang, T. N. Fu, C. F. Li, *Painterly Rendering with Content-dependent Natural Paint Strokes*, The Visual Computer: International Journal of Computer Graphics **27** (2011), 861–871.

[16] T. Kamada, S. Kawai, *An Enhanced Treatment of Hidden Lines*, ACM Transactions on Graphics (TOG) **6** (1987), 308–323.

[17] V. Karnati, M. Uliyar, S. Deyl, *Fast Non–Local Algorithm for Image Denoising*, ICIP '09 Proceedings of the16th IEEE International Conference on Image Processing, IEEE Press Piscataway NJ, USA, 2009, str. 3829–3832.

[18] A. Klein, M. M. Kazhdan W. Li, W. T. Corrêa, A. Finkelstein, T. A. Funkhouser, *Non-photorealistic Virtual Environments*, SIGGRAPH '00 Proceedings of the 27th annual conference on Computer graphics and interactive techniques, ACM Press/Addison-Wesley Publishing Co. New York, NY, USA, 2000, str. 527–534.

[19] H. Lee, C. H. Lee, K. Yoon, *Motion Based Painterly Rendering*, EGSR'09 Proceedings of the Twentieth Eurographics conference on Rendering, Eurographics Association Aire-la-Ville, Switzerland, Switzerland, 2009, str. 1207–1215.

[20] M. Lindenbaum, M. Fischer, A. M. Bruckstein, *On Gabor's Contribution to Image Enhancement*, Pattern Recognition **27** (1994), 1–8.

[21] P. Litwinowicz, *Processing Images and Video For an Impressionist Effect*, SIGGRAPH '97 Proceedings of the 24th annual conference on Computer graphics and interactive techniques, ACM Press/Addison-Wesley Publishing Co. New York, NY, USA, 1997, str. 407–414.

[22] C. Lu, L. Xu, J. Jia, *Combining Sketch and Tone for Pencil Drawing Production*, NPAR '12 Proceedings of the Symposium on Non-Photorealistic Animation and Rendering, Eurographics Association Aire-la-Ville, Switzerland, 2012, str. 65–73.

[23] Y. Yamaguchi M. Shiraishi, *An Algorithm for Automatic Painterly Rendering Based on Local Source Image Approximation*, NPAR '00 Proceedings of the 1st international symposium on Non-photorealistic animation and rendering, ACM New York, NY, USA, 2000, str. 53–58.

[24] B. J. Meier, *Painterly Rendering for Animation*, SIGGRAPH '96 Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, ACM New York, NY, USA, 1996, str. 477–484.

[25] K. Miyata, *A method of generating stone wall patterns*, SIGGRAPH '90 Proceedings of the 17th annual conference on Computer graphics and interactive techniques, ACM New York, NY, USA, 1990, str. 387–394.

[26] M. L. Morbey, *From Canvas to Computer: Harold Cohen's Artificial Intelligence Paradigm for Art Making*, doktorska disertacija, The Ohio State University, Ohio, USA, 1992.

[27] P. Perona, J. Malik, *Scale-Space and Edge Detection Using Anisotropic Diffusion*, IEEE Transactions on Pattern Analysis and Machine Intelligence **12** (1990), 629–639.

[28] D. Rudolf, D. Mould, E. Neufeld, *Simulating Wax Crayons*, PG '03 Proceedings of the 11th Pacific Conference on Computer Graphics and Applications, IEEE Computer Society Washington, DC, USA, 2003, str. 163–172.

[29] D. Rudolf, D. Mould, E. Neufeld, *A Bidirectional Deposition Model of Wax Crayons*, Computer Graphics Forum **24** (2005), 27–39.

[30] M. P. Salisbury, M. T. Wong, J. F. Hughes, D. H. Salesin, *Orinetable Textures for Image-Based Pen-and-Ink Illustration*, SIGGRAPH '97 Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, ACM Press/Addison-Wesley Publishing Co. New York, NY, USA, 1997, str. 401–406.

[31] D. D. Seligmann, S. Feiner, *Automated Generation of Intent-based 3D Illustrations*, SIGGRAPH '91 Proceedings of the 18th annual conference on Computer graphics and interactive techniques, ACM New York, NY, USA, 1991, str. 123–132.

[32] S. M. Smith, J. M. Brady, *SUSAN - A New Approach to Low Level Image Processing*, International Journal of Computer Vision **23** (1997), 45–78.

[33] A. Hertzmann T. Goodwin, I. Vollick, *Isophote Distance: A Shading Approach to Artistic Stroke Thickness*, NPAR '07 Proceedings of the 5th international symposium on Non-photorealistic animation and rendering, ACM New York, NY, USA, 2007, str. 53–62.

[34] S. M. F. Treavett, M. Chen, *Statistical Techniques for the Automated Synthesis of Non-photorealistic Images*, Proc. 15th Eurographics UK Conference, 1997.

[35] G. Turk, D. Banks, *Image-guided Streamline Placement*, SIGGRAPH '96 Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, ACM New York, NY, USA, 1996, str. 453–460.

[36] G. Winkenbach, D. H. Salesin, *Computer-Generated Pen-and-Ink Illustrations*, SIGGRAPH '94 Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques, ACM New York, NY, USA, 1994, str. 91–100.

[37] G. Winkenbach, D. H. Salesin, *Rendering Parametric Surfaces in Pen and Ink*, SIGGRAPH '96 Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, ACM New York, NY, USA, 1996, str. 469–476.

[38] L. P. Yaroslavsky, *Digital Picture Processing*, Springer-Verlag New York, Inc. Secaucus, NJ, USA, 1985.