TARTU ÜLIKOOL
MATEMAATIKA-INFORMAATIKATEADUSKOND
Arvutiteaduse instituut
Informaatika eriala

**Ville Sokk**

# Tekstuurisüntees rastergraafikaredaktorile GIMP

**Bakalaureusetöö (6 EAP)**

Juhendaja: vanemteadur Sven Laur

Autor: ..... “.....“ mai 2013
Juhendaja: ..... “.....“ mai 2013

Lubada kaitsmisele

Professor: ..... “.....“ mai 2013

TARTU 2013

# Sisukord

# 1. Sissejuhatus

## 1.1. Tekstuurisüntees, GIMP ja GEGL

Tekstuur tähendab digitaalset pilti, mis kantakse mingile geomeetrilisele objektile, et tekitada selle pinnale värvi, läbipaistvuse, peegelduvuse või millegi muu variatsiooni [1]. Tekstuure läheb tarvis 3D graafikas filmide ja videomängude jaoks modelleeritud objektide realistlikumalt või kunstipärasemalt esitamiseks. Tihtipeale kasutatakse tekstuuri kitsamat definitsiooni — pilt, mis koosneb mingist korduvast mustrist. Muster võib olla nii loomulik (näiteks muru) kui tehislik (kivisein). Muster ei pea olema korrapärane. Tekstuure saab luua käsitsi, fotode põhjal rastegraafikaredaktoris või genereerida algoritmi abil. Edaspidi kasutatakse tekstuuri kitsamat tähendust.

Tekstuurisüntees (inglise keeles *texture synthesis*) on meetod, mis lihtsustab tekstuuride loomise protsessi. Tekstuurisünteesi algoritmi eesmärki võib kirjeldada järgnevalt: antud tekstuuri näidise põhjal luua uus (kasutaja soovitud mõõtmetega) tekstuur, mis näib olevat genereeritud sama protsessi poolt [1]. On olemas algoritme, mis ei vaja etteantud pilti (näiteks Perlini müra), aga need ei vasta sellele definitsioonile ja edaspidi nendega töös ei tegeleta.

Tekstuurisünteesi algoritmid on osutunud kasulikuks ka mujal kui 3D graafikas, sealjuures ka juhtudel kus tegeletakse piltidega, mis ei koosne ilmtingimata mustrist. Näiteks on tekstuurisünteesi rakendatud järgmiste probleemide lahendamisel:

1) pildilt objektide eemaldamine
2) pildile objekti kleepimine teiselt pildilt
3) pildil objektide ümberpaigutamine
4) pildi mõõtmete muutmine säilitades objektide proportsioonid
5) pildi suurendamine genereerides puuduvat informatsiooni
6) pildi teravustamine
7) pildifailide pakkimine
8) staatilise pildi animeerimine

GIMP (*GNU image manipulation program*) on avatud lähtekoodiga rastergraafikaredaktor, mida võib kasutada fotode töötlemiseks, digitaalmaalimiseks, piltide konverteerimiseks erinevate formaatide vahel, kasutajaliideste graafiliste elementide disainimiseks, astrofotograafiaks jms jaoks. GIMP toetab pistikprogrammide ja skriptide kirjutamist (programmeerimiskeeltes Scheme, Python, Perl, Ruby, C), mis võimaldavad kasutajatel oma tööprotsessi mugavdada, GIMPi funktsionaalsust täiendada või kasutada GIMPi uute algoritmide katsetamiseks.

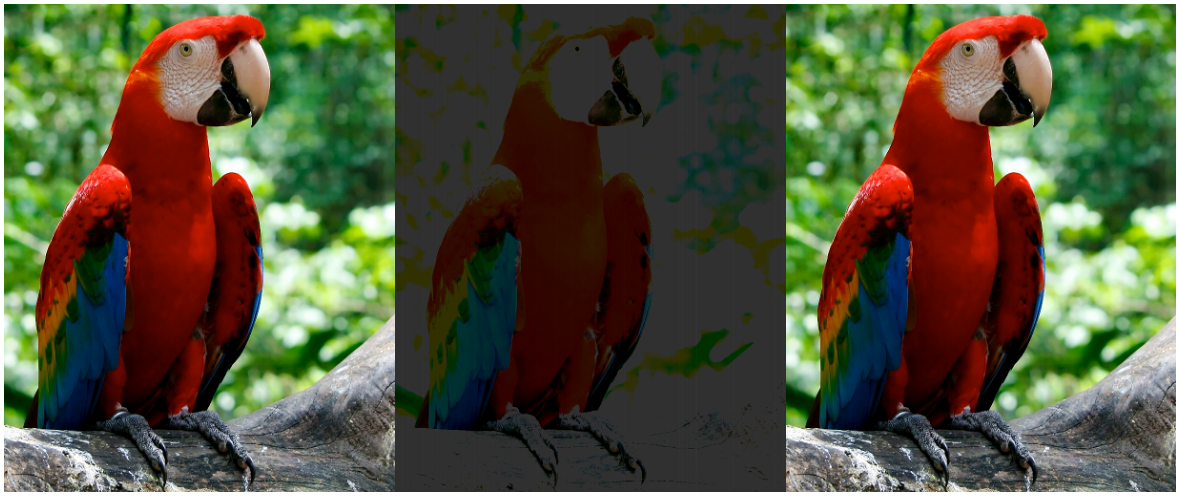
GEGL (*generic graphics library*) on graafidel põhinev pilditöötluse raamistik. Kui enamikes pilditöötluse teekides ja rakendustes muudab programmeerija teegi funktsioonide abil pildi puhvrit, siis GEGLis koostatakse erinevatest pilditöötluse operaatoritest tsükliteta graaf, kus mingid tipud on sisendid (näiteks salvestusseadmelt laetud pildid) ja mingi tipp väljund (näiteks aken). See võimaldab paindlikumalt luua programme, kus kasutajatel on vaja palju katsetada, sest igal ajahetkel on võimalik muuta varem graafi lisatud operaatorite parameetreid, ega pea kasutama sammude tagasivõtmist ja sellega kaotsi läinud tööd uuesti tegema. GEGL kasutab omakorda teeki babl, mis võimaldab konverteerida erinevate värvide esituste vahel.

## 1.2. Ülesandepüstitus

Rastergraafikas on pilt esitatud kahedimensioonilise pikslite massiivina. Iga piksli värvus on omakorda esitatud arvude jadana, mis on koordinaadid värviruumis. Värviruum on täpne kirjeldus koordinaatide süsteemist ja alamruumist selles süsteemis, kus iga punkt on üks värv [2, lk 289]. Lisaks värviruumile on oluline bitisügavus ehk millist arvutüüpi koordinaatide esituseks kasutada. Mida suurem bitisügavus, seda rohkem värve on võimalik esitada. Joonisel 1 on foto RGB värviruumis 8- ja 2-bitise sügavusega. 2-bitise pildi puhul on märgata, et võimalikke värve on liiga vähe. GIMP toetab väheseid värvide esituse viise (nii värviruumide kui bitisügavuse seisukohalt) ja GIMPi on seetõttu palju kritiseeritud. GIMPis töötatakse enamasti 8-bitiste RGBA värvidega (koordinaadid on punane, roheline, sinine ja läbipaistvus, neist igaüks on 8-bitine märgita täisarv). Kuna 8-bitiste märgita täisarvude ulatus on väike (0-255), siis pilte töödeldes toimub informatsiooni kadu. Tagajärjena tekivad pildil defektid (seda illustreerib joonis 2). Samuti esineb informatsiooni kadu, kui avada suurema bitisügavusega pilt (näiteks professionaalse kaameraga tehtud foto). Olukorra parandamiseks loodigi GEGL, millest eraldus hiljem

teek babl. GIMPi arendajate eesmärk on versiooniks 2.10 asendada GIMPi olemasolev pilditöötluse funktsionaalsus GEGLiga.

Joonis 1. Bitisügavuse näide. Vasemal pildil on bitisügavus 8 ja paremal 2. Papagoi foto pärineb allikast [3].



Joonis 2. Näide defektist, mis võib tekkida 8-bitiseid pilte töödeldes. Vasakul on esialgne pilt. Keskel on pilti 8-bitisena töödeldes palju heledamaks tehtud ja siis püütud esialgsele tasemele tagasi tuua. Paremal on pilt viidud 32-bitiseks ja siis sama protsess läbitud.

GIMPi jaoks on loodud mitu tekstuurisünteesi pistikprogrammi, millel on aga mitu puudust. Töö eesmärk on programmeerida GIMPile tekstuurisünteesi algoritmi implementatsioon, mis lahendaks need probleemid. Kuna GIMP hakkab kasutama GEGLit, siis tuleb see programmeerida GEGLi täiendusena ning seejärel ühendada GIMPiga. Esmane eesmärk on programmeerida algoritmi põhifunktsionaalsus ja siis selle põhjal mõni tekstuurisünteesi lisavõimalus.

## 1.3. Kirjanduse ülevaade

Tekstuurisünteesi algoritme võib jagada kaheks.

1. Inimese nägemistaju modelleerivad matemaatilised mudelid, mis analüüsivad näitepilti, et leida seda iseloomustavad parameetrid ning sünteesivad nende põhjal uue pildi.
2. Tõenäosusteooriast pärineva *Markov random field*'i (edaspidi MRF) ideedel baseeruvad algoritmid.

Teise kategooria algoritme on rohkem, need on lihtsamad ja on seni andnud paremaid tulemusi. Seetõttu tegeletakse töös edaspidi vaid nendega.

MRF meetodid käsitlevad tekstuure kui juhusliku protsessi poolt tekitatud objekte. Sellel protsessil on kaks olulist omadust [4].

1. Iga piksli väärtus on määratud teda teatud suurusega fikseeritud alal ümbritsevate pikslite poolt.
2. Piksli väärtus ei sõltu asukohast tekstuuris.

MRF põhjal võib tekstuurisünteesi eesmärki sõnastada nii: antud lähtepildi põhjal sünteesida tekstuur nii, et tekstuuri iga piksli ümbrus on sarnane vähemalt ühele ümbrusele lähtepildis [1]. Kui lähtepilt vastab MRF-i kahele eeldusele, siis ümbruste sarnasus tagab ka tekstuuri ja lähtepildi sarnasuse [1].

On olemas algoritme, mis baseeruvad MRF mudelitel, mitte lihtsalt sarnaste ümbruste ideel, aga need on keerulised ja aeglased [1].

Efrose ja Leungi [5] algoritm kasutab ainult MRF-i eeldusi ja sellest lähtuvat sarnaste ümbruste ideed. Väljund genereeritakse ühe piksli kaupa. Piksli värvuse otsustamiseks vaadatakse tema ümbruses juba genereeritud piksleid, otsitakse lähtepildist sarnaseid ümbrusi, valitakse juhuslikult neist üks, milles vastaval kohal oleva piksli värvus saab sünteesitava piksli värvuseks. Töö alustuseks kopeeritakse sisendist väljundisse mingisugune väike ala. Sünteesimisel kasutatava ümbruse mõõtmed saab valida kasutaja. Kuna see algoritm on väga lihtne ja heade tulemustega, on see saanud paljude teiste tööde aluseks. Algoritmil on kaks probleemi: see on aeglane (ümbruste otsimisel lähtepildist on

liiga palju võimalusi ja suurte ümbruste puhul on erinevuse arvutamine aeglane) ja genereerib mõnikord halba väljundit (kopeerib korduvalt sisendit või produtseerib müra). Väljundi kvaliteeti mõjutab kasutaja valitud ümbruse suurus ja pikslite sünteesimise järjekord. Algoritmide kohta, mis sünteesivad piksleid ühe kaupa, öeldakse inglise keeles *pixel-based synthesis* ja neid võib pidada MRF-i ideedest lähtuvate meetodite üheks haruks.

Teine haru on *patch-based synthesis*. Kui eelmisel juhul kopeeriti piksleid sisendist väljundisse, siis siin kopeeritakse terveid piirkondi. *Patch-based synthesis*'e eelised on kiirus ja kvaliteet. Probleemiks on väljundisse kleebitud piirkondade vaheliste piiride märgatavus. Algoritm *image quilting* [6] lahendab probleemi järgnevalt: väljundisse kopeeritakse ruute, mis väljundis teatud ulatuses kattuvad. Kattuvas alas leitakse optimeerimise teel serv, millest ühele poole jäävad pikslid võetakse ühest ruudust ja teisele poole jäävad teisest ruudust. Optimeeritav funktsioon on defineeritud nii, et serv kulgeks sarnaste pikslite vahelt, seega on leitud serv võimalikult vähe märgatav.

Kwatra et al. [7] algoritm, mida nimetatakse tekstuuri optimeerimiseks (inglise keeles *texture optimization*), sarnaneb *pixel-based synthesis*'ele (sünteesi ühik on piksel), aga igat väljundi pikslit mõjutavad kõik teised pikslid, mitte ainult varem sünteesitud pikslid. Optimeerimine on iteratiivne. Iteratsioon koosneb kahest sammust: E-samm ja M-samm (nimetused tulevad sellest, et see sarnaneb statistikast pärinevale *expectation-maximization*'i algoritmile). M-sammus otsitakse väljundi pikslitele sarnased ümbrused lähtepildist (nagu *pixel-based synthesis'*e puhul) ja E-sammus muudetakse tekstuuri vastavalt M-sammu tulemustele. Samme korrates muutub väljund lähtepildile sarnasemaks.

*Tile-based synthesis* meetodid koostavad hulga plaate ehk geomeetrilisi kujundeid, millega saab tasandi ilma tühimiketa katta. Etteantud tekstuuri põhjal koostatakse selliste tekstuuriga plaatide hulk, et sobivaid plaate kõrvuti asetades ei jää plaatide vahelised servad näha. Näiteks artiklis [8] kasutatakse kolmnurkseid plaate, et sünteesida tekstuur otse geomeetrilise objekti pinnale, ilma, et kunstnik peaks looma pinnalaotuse.

# 2. Algoritm

## 2.1. Algoritmi ülevaade

Selles peatükis kirjeldatud osa algoritmist põhineb artiklil [7]. See on tekstuurisünteesi meetod, mida nimetatakse tekstuuri optimeerimiseks (inglise keeles *texture optimization*). Tavalised *pixel-based* algoritmid sünteesivad tekstuuri ühe piksli kaupa valides piksli ümbruse põhjal lähtepildist värvi. Väljundi kvaliteedi seisukohalt on sellel lähenemisel kaks probleemi. Kui algoritm teeb mingi piksli sünteesimisel halva valiku, siis võib see edasise sünteesi ära rikkuda. Teine probleem on, et sünteesitud piksel ei mõjuta enne teda sünteesitud pikslite värvust. Tekstuuri optimeerimine loob tekstuuri esialgse lähendi ja parandab seda iteratiivselt. Selleks on defineeritud funktsioon, mida nimetatakse energiaks ja mis hindab väljundi ja lähtepildi erinevust. Algoritm vähendab energiat ja muudab sellega väljundit sisendile sarnasemaks.

MRF-il põhinevate meetodite eesmärk oli sünteesida tekstuur, kus iga piksli ümbrusel leidub sarnane ümbrus lähtepildis. Selle saavutamiseks on algoritmides üheks sammuks väljundi ümbrustele sarnaste lähtepildi ümbruste otsimine. Otsimisel on vaja hinnata ümbruste sarnasust. Olgu $X$ lähtepilt ja $Y$ sünteesitav pilt. Piksli $p \in Y$ ümbrust kujutletakse vektorina $y_p$, kus ümbruses asuvate pikslite koordinaadid paiknevad järjestikku, ehk RGB värviruumi ja $w \times w$ mõõtmetega ümbruste korral

$$y_p = (R_1, G_1, B_1, R_2, G_2, B_2, \ldots, R_{w^2}, G_{w^2}, B_{w^2}),$$

kus indeksiga $x$ on tähistatud ümbruse $x$-nda piksli koordinaadid. Olgu $x_p$ lähtepildi ümbruse, mida soovitakse $y_p$-ga võrrelda, vektor. Ümbruste erinevus on vektorite vastavate komponentide vahede ruutude summa ehk vektorite kauguse ruut $\|x_p - y_p\|^2$.

Oletame, et sünteesitud tekstuuri $Y$ iga piksli $p$ ümbrusele $y_p$ on leitud sarnane ümbrus $x_p$ lähtepildis. Tekstuuri energia funktsioon on

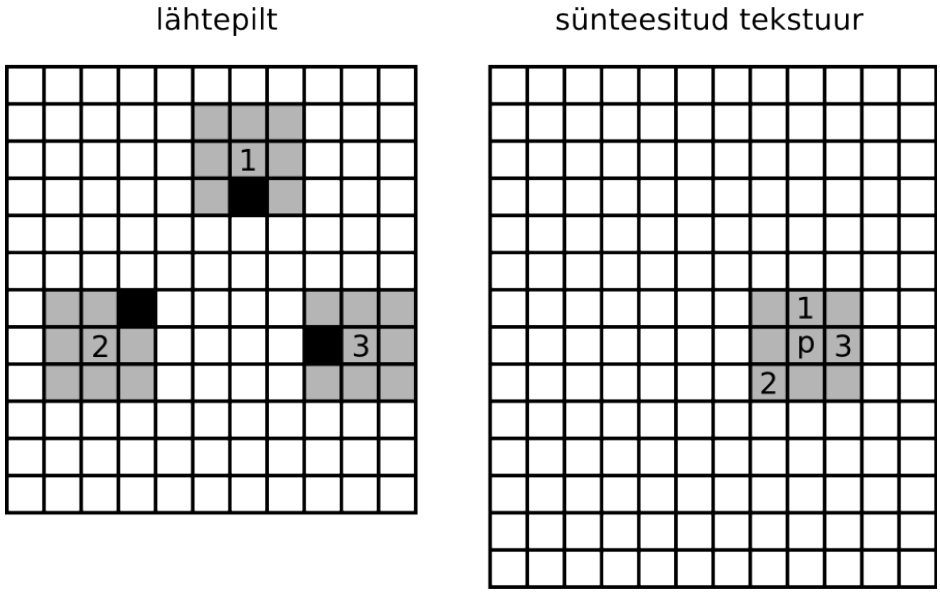
$$E(Y) = \sum_{p \in Y} \|y_p - x_p\|^2.$$

Optimeeriva algoritmi eesmärk on muuta tekstuuri nii, et selle funktsiooni väärtus väheneks (ehk lähtepildi ja tekstuuri tajutav erinevus väheneks). Selleks kasutatakse

algoritmi, mis sarnaneb statistikast pärinevale algoritmile *expectation-maximization*. See koosneb kahest sammust, mida nimetatakse edaspidi M-sammuks ja E-sammuks.

M-sammus leitakse iga piksli ümbrusele võimalikult sarnane ümbrus (vaste) lähtepildis ja jäetakse meelde selle koordinaadid. Olgu piksli $p \in Y$ ümbruse vektor $y$. Kõige sarnasem on lähtepildi ümbrus, mille vektori $x$ korral $\|x-y\|^2$ on kõige väiksem.

E-sammus muudetakse tekstuuri kõiki piksleid M-sammus leitud vastete põhjal. Seda selgitab joonis 3. Kui sünteesimisel kasutatakse $3\times3$ ümbrusi ja hetkel muudetakse pikslit $p$, siis sünteesitud tekstuuris olevad hallid pikslid paiknevad $p$ ümbruses. Näitena on toodud pikslite 1, 2, 3 vasted lähtepildis. Iga $i \in \{1,2,3\}$ korral on vaste ümbruses must piksel, mis on nihutatud nii, et see paikneks vaste keskpunkti suhtes nagu $p$ paikneb $i$ suhtes tekstuuris. Mustad pikslid on hääled. Kõikide $p$ ümbruses asuvate pikslite põhjal niimoodi leitud häälte aritmeetiline keskmine on $p$ uus väärtus. $w\times w$ mõõtmetega ümbruste korral on seega $w^2$ häält selle kohta, milline peaks $p$ olema, et väljund sarnaneks lähtepildile.




Joonis 3. E-sammus piksli *p* uuendamine pikslite 1, 2, 3 häälte järgi.

E- ja M-samm moodustavad ühe iteratsiooni. Itereerimise võib peatada, kui kahe järjestikuse M-sammu leitud vasted on samad (sel juhul väljund enam ei muutu) või kui iteratsioonide arv ületab mingi fikseeritud piiri. Esialgses algoritmis alustati E-sammust ja

selleks määrati enne optimeerimist sünteesitava tekstuuri ümbrustele suvalised vasted lähtepildis. Käesolevas töös läbitakse esimesena M-samm ja seetõttu kopeeritakse enne optimeerimist juhuslikelt koordinaatidelt piksleid sisendist väljundisse. Nii saadakse sünteesitava tekstuuri esialgne lähend.

Intuitiivselt võib algoritmi toimimise põhjust selgitada järgnevalt. MRF-il põhinevate meetodite eesmärk oli sünteesida tekstuur, mille ümbrustel leiduvad sarnased lähtepildi ümbrused [1]. See tagab lähtepildi ja tekstuuri sarnasuse. M-sammus otsitaksegi tekstuuri ümbrustele vasteid lähtepildist. E-sammus lahendatakse kattuvate ümbruste vahel tekkivat konflikti. See võimaldab M-sammul järgmises iteratsioonis leida veel sarnasemad vasted.

Kuna M-sammule kulub enamus algoritmi tööajast, soovitavad autorid otsida vasteid ümbrustele, mille keskpunktid asuvad $w/4$ vahega ruudustikul, kus $w$ on ümbruse külje pikkus. Sellisel juhul on E-sammus piisavalt hääletajaid (iga piksli ümbruses on piisavalt piksleid, millele on leitud vasted), aga M-sammus kulutatakse ümbruste otsimisele vähem aega.

Paljudes algoritmides jäetakse ümbruse suurus kasutaja valida. Tekstuuri optimeerimise autorid soovitavad kasutada mitme suurusega ümbrusi alustades suuremast. See võimaldab algoritmil sünteesida kõigepealt suuremaid ja siis väiksemaid detaile. Käesolevas töös loodud lahenduses kasutatakse kahe suurusega ümbrusi — $7\times7$ ja $15\times15$. Neist viimast kasutatakse esimesel kümnel iteratsioonil.

Sellel algoritmil on aga mitu olulist probleemi.

- M-samm on aeglane, sest kõikide võimaluste läbiproovimisel on iga piksli ümbruse vastele väga palju kandidaate.
- Mida suuremad ümbruse mõõtmed valime, seda aeglasemalt algoritm töötab. Ümbruse suurus mõjutab aga oluliselt väljundi kvaliteeti — mida suuremad on lähtepildis detailid, seda suurem peab olema ümbrus, et detailid ka väljundisse kanduksid.
- Kui E-sammus kasutatakse piksli uue väärtuse leidmiseks aritmeetilist keskmist, siis on tulemus udune ning ei pruugi lähtepildile sarnaneda.

Probleemide lahendusi on kirjeldatud järgnevates peatükkides.

## 2.2. Püramiid

Püramiid (inglise keeles *image pyramid* või *pyramid* või *pyramid representation*) on järjend mingist pildist ja tema vähendatud suurusega koopiatest. Kui koopia *a* on väiksema suurusega kui *b*, siis edaspidi öeldakse, et *a* on püramiidis kõrgemal kui *b*.

Püramiidi võib vastavalt vajadusele koostada kas väikest pilti suurendades või suurt pilti vähendades. Käesolevas töös on vähendamise protsess järgnev:

1) siluda pilt kasutades konvolutsiooni $3\times3$ Gaussi filtriga;
2) eemaldada pildist iga teine piksel nii *x* kui *y* suunas.

Suurendamise protsess on vastupidine:

1) igast pikslist teha neli kõrvuti asetsevat pikslit (üks $2\times2$ ruut, mis on piksli värvi);
2) siluda pilt kasutades konvolutsiooni $3\times3$ Gaussi filtriga.

Sellisel viisil suurendades/vähendades saadud püramiidi nimetatakse inglise keeles *Gaussian pyramid*'iks.

Konvolutsioon on pilditöötlusoperatsioon, mida kasutatakse näiteks teravustamiseks või servade tuvastamiseks. Konvolutsiooni tarbeks on vaja defineerida $m\times n$ maatriks (*m*, *n* on paaritud arvud), mida nimetatakse filtriks. Kokkuleppeliselt on filtri keskmise elemendi koordinaadid $(0,0)$. Olgu mingis pildis $f$ kohal $(x,y)$ asuva piksli väärtus antud funktsiooniga $f(x,y)$. Rakendades pildile $f$ konvolutsiooni $m\times n$ filtriga *M*, saab pildi, mille pikslite väärtused on antud valemiga

$$g(x,y)=\sum_{s=-a}^{a}\sum_{t=-b}^{b} m_{s,t} f(x+s,y+t),$$

kus $a=(m-1)/2$, $b=(n-1)/2$ (valemi allikas on [2, lk 117]). Ehk siis piksli väärtus väljundis on tema naabruses asuvate pikslite ja vastavate filtri elementide korrutiste summa.

$3\times3$ Gaussi filter on

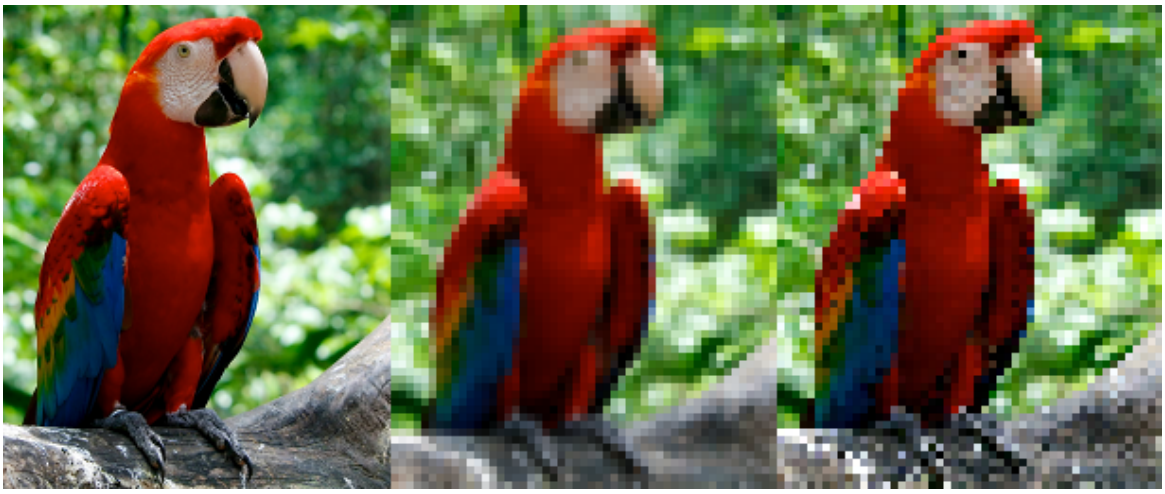
$$\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}/16.$$

Nimetus tuleneb sellest, et filtri element kohal $i, j$ on lähedane normaaljaotuse tihedusfunktsioonist saadud funktsiooni

$$G(x,y)=\frac{1}{2\pi\sigma^2}e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

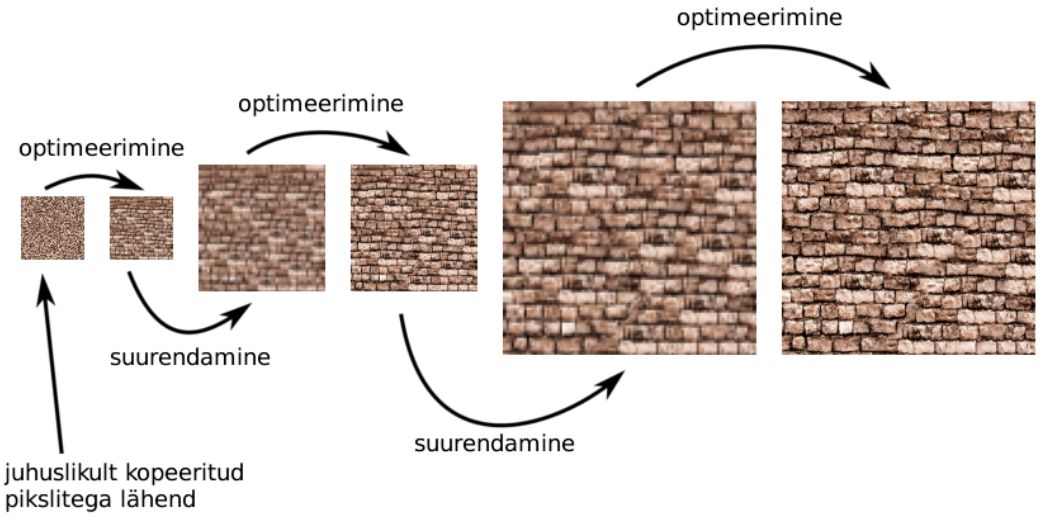
väärtusele kohal $G(i,j)$, kui $\sigma\approx 0.8$ (valemi allikas on [9]). Funktsiooniga $G$ saab silumise tarbeks koostada ka suuremaid filtreid, aga käesolevas töös kasutatakse $3\times 3$ filtrit.

Intuitiivselt on silumise eesmärk levitada pikslis sisalduvat informatsiooni kõrvalasuvatele pikslitele. Kui näiteks vähendamisel piirduda ainult teise sammuga, siis tekivad sakilised servad, müra või mustrid, mida esialgsel pildil pole. Seda illustreerib joonis 4, kus alustades vasakult on toodud esialgne pilt, kirjeldatud meetodil vähendatud pilt ja ainult sammu (2) kasutades vähendatud pilt.



Joonis 4. Pildi vähendamise meetodite võrdlus. Papagoi foto pärineb allikast [3].

Püramiidi kasutamine võimaldab algoritmi tööd oluliselt kiirendada. Lähtepildist koostatakse püramiid (käesoleva töö lahenduses valitakse kõrgus nii, et kõrgeima taseme lühema külje pikkus oleks ligikaudu 32) ja sünteesima hakatakse mitu korda väiksemate mõõtmetega tekstuuri. Kuna lähtepildi kõrgeim tase ja sünteesitav tekstuur on väga väikesed, siis nii E- kui M-samm võtavad oluliselt vähem aega. Kui süntees on lõppenud, siis tekstuuri suurendatakse ja minnakse lähtepildi püramiidis tase madalamale. Sellel tasemel kulgeb optimeerimine kiiremini, sest tekstuuri suurendamisel saadud pilt on hea lähend. Nii jätkatakse kuni madalaima taseme tekstuuri süntees on lõppenud. Seda illustreerib joonis 5.




Joonis 5. Tekstuuri optimeerimine kasutades püramiidi.

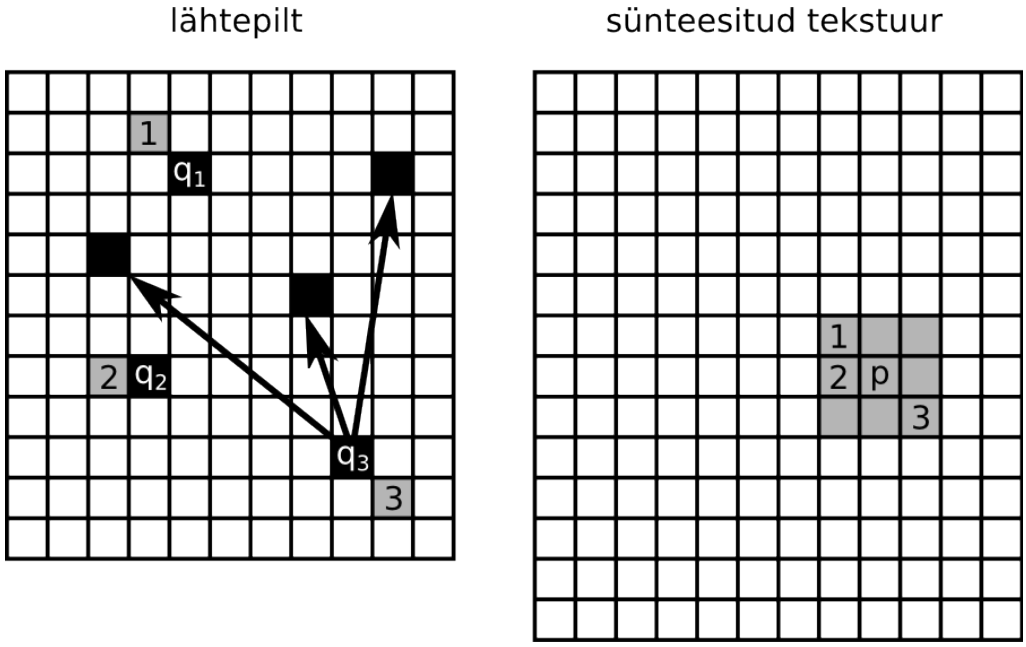
Ühtlasi võimaldab püramiid kasutada väiksemat ümbrust. Kuna kõrgemal tasemel on detailid väiksemad, siis ümbrus katab rohkem detaile kui sama suurusega ümbrus madalal tasemel. Suuremad detailid saavad paika kõrgematel tasemetel ja seetõttu võib kasutada väiksemat ümbrust.

## 2.3. K-koherentsus

Artiklis [10] kasutatakse M-sammu kiirendamiseks algoritmi k-koherentsus (inglise keeles *k-coherence)*, mis ilmus esmakordselt artiklis [11]. K-koherentsus lähtub tähelepanekust, et lähtepildis lähestikku paiknevad pikslid asuvad ka sünteesitud tekstuuris lähestikku. K-koherentsust kasutades kopeeritakse piksleid lähtepildist väljundisse ja peetakse meeles lähtepildi piksli koordinaadid. See võimaldab vähendada M-sammus võrreldavate ümbruste arvu.

Enne sünteesimist toimub lähtepildi eeltöötlus. Selle käigus otsitakse lähtepildi pikslite ümbrustele vasteid lähtepildist endast. Iga piksli puhul jäetakse meelde *k*-1 parima vaste keskpunkti koordinaadid. Parameeter *k* mõjutab sünteesi kvaliteeti ja jäetakse kasutaja valida, tavaliselt vahemikust 2 kuni 11.

K-koherentsusega M-sammu illustreerib joonis 6, kus $k=4$ ja $w=3$. M-sammus otsitakse jätkuvalt kõikide pikslite ümbrustele vaste. Joonisel otsitakse vastet piksli *p* ümbrusele. Kõigepealt koostatakse kandidaatide hulk $C(p)$. Halliga on joonisel tähistatud *p* ümbruses asuvad pikslid. Näitena on välja toodud pikslite 1, 2, 3 ümbruste vasted lähtepildis. Vastete ümbruses on pikslid $q_1, q_2, q_3$. Iga $i \in \{1,2,3\}$ korral on $q_i$ nihutatud nii, et $q_i$ paikneb vaste keskpunkti suhtes nagu *p* paikneb *i* suhtes tekstuuris. Iga $q_i$ lisatakse hulka $C(p)$. Iga $q_i$ ümbrusele on eeltöötluse käigus leitud *k*-1 vastet lähtepildist endast. Näiteks on joonisel nooltega tähistatud $q_3$ vasted. Need lisatakse samuti hulka $C(p)$. Piksli *p* ümbruse vasteks lähtepildis valitakse see $x \in C(p)$, mille korral on *x* ja *p* ümbruste erinevus kõige väiksem. Näites koostati kandidaatide hulk pikslite 1, 2, 3 järgi, aga reaalselt koostatakse see kõigi *p* ümbruses asuvate pikslite järgi.




Joonis 6. K-koherentsusega pikslile *p* vaste otsimine.

Kuna k-koherentsus nõuab lisaks pikslite värvustele ka koordinaatide talletamist, siis tuleb seda kasutades muuta ka E-sammu. Esialgses E-sammus leiti M-sammu vastete järgi hääled ning määrati piksli värvuseks nende aritmeetiline keskmine. Sellist pikslit ei pruugi lähtepildis esineda ja kui esinebki, siis pole selle asukoht teada. Muudetud E-sammus arvutatakse keskmine ning leitakse siis häälte seast piksel, mis on keskmisele kõige lähedasem, asendatakse sünteesitav piksel sellega ning talletatakse selle koordinaadid.

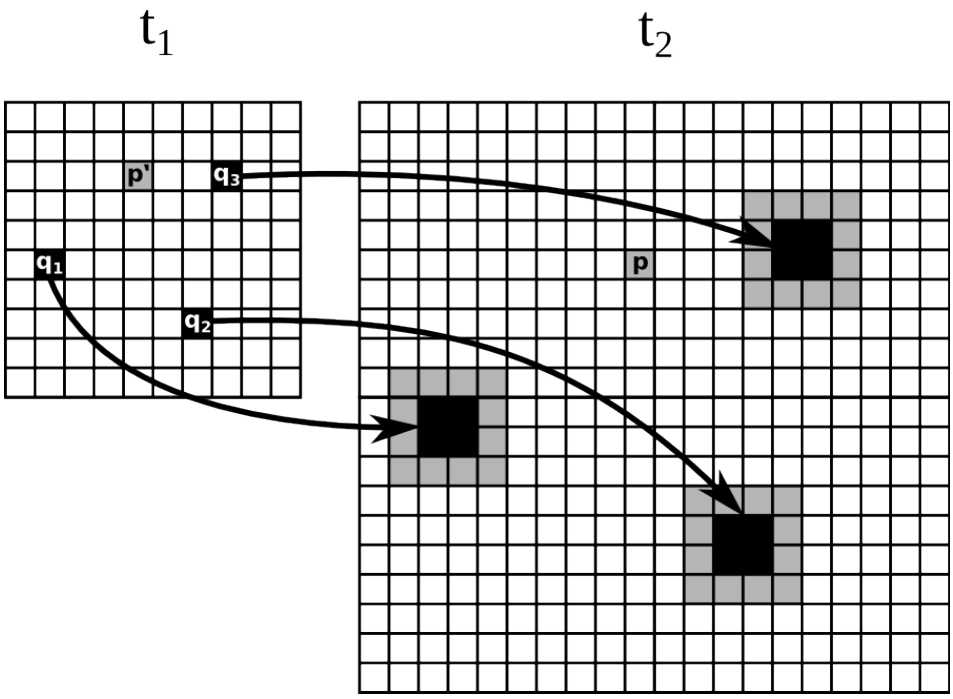
K-koherentsus vähendab oluliselt otsingu käigus võrreldavate ümbruste arvu ja seega kiirendab algoritmi tööd. Kuigi see ei taga parimate vastete leidmist on leitud vasted piisavalt head, et sünteesida hea kvaliteediga tekstuur. Kuna piksleid kopeeritakse sisendist väljundisse, mitte ei kasutata keskmist, siis tekib tekstuuri udusust oluliselt vähem.

Kasutades püramiidi, on vaja eeltöötluse samm läbida kõigil püramiidi tasemetel. Artikkel [11] kirjeldab selleks meetodit, kus kõrgeimal tasemel kasutatakse jõumeetodil otsingut suure *k* väärtusega (see on kiire, sest kõrgeima taseme pilt on väike) ja madalamatel tasemetel kasutatakse kõrgeimal tasemel leitud vasteid kiireks, aga mitte enam ammendavaks otsinguks. Käesolevas töös on eeltöötluseks kasutatud teistsugust algoritmi, mida kirjeldatakse järgmises peatükis.

## 2.4. Otsingu eeltöötlus

Otsingu eeltöötlus põhineb algoritmil *mixed-resolution patch matching* [12] (edaspidi MRPM). MRPM on loodud, et leida ühe pildi ümbrustele vasteid teisest pildist. Käesolevas töös kasutatakse MRPM-i k-koherentsuse eeltöötluseks ehk selleks, et leida vasteid lähtepildi ümbrustele lähtepildist.

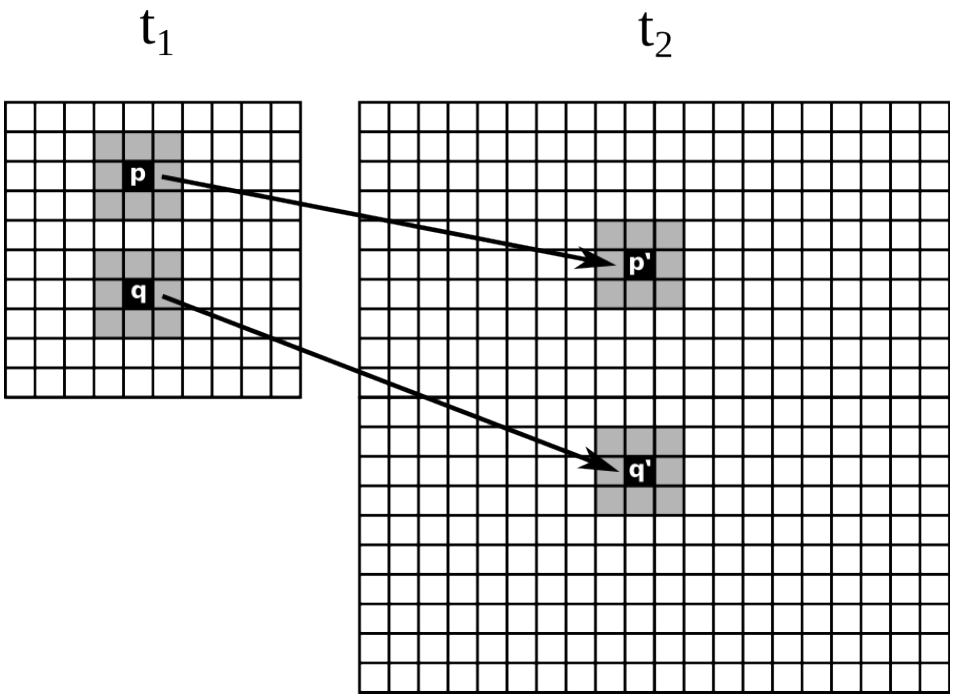
MRPM kasutab otsingu kiirendamiseks püramiidi. Kõrgeimal tasemel kasutatakse jõumeetodil otsingut. Algoritmi tööd madalamatel tasemetel illustreerib joonis 7. $t_2$ on tase, mille ümbrustele otsitakse vasteid ja $t_1$ vahetult kõrgem tase. Näites otsitakse vastet piksli *p* ümbrusele. Kõigepealt tuleb koostada kandidaatide hulk $C(p)$. Olgu $(x, y)$ *p* koordinaadid tasemel $t_2$. Pikslile *p* vastab taseme $t_1$ piksel *p′* koordinaatidega $(\lfloor x/2 \rfloor, \lfloor y/2 \rfloor)$. Tasemel $t_1$ on juba kõikidele ümbrustele *k*-1 vastet leitud. Joonisel on $k=4$ ja *p′* ümbruse vastete keskpunktide tähised on $q_1, q_2, q_3$. Igale pikslile $q_i$ koordinaatidega $(x, y)$ vastab tasemel $t_2$ neli pikslit koordinaatidega $(2x, 2y), (2x+1, 2y), (2x, 2y+1), (2x+1, 2y+1)$ (joonisel näidatud noolega). Nende pikslite $u \times u$ ümbrustesse (joonisel $u=3$, töös loodud lahenduses $u=9$ ) jäävad pikslid (joonisel hallid ja mustad) kuuluvad hulka $C(p)$. Pikslile *p* valitakse hulgast $C(p)$ *k*-1 vastet, mille ümbruse erinevus *p* ümbrusest on kõige väiksem.




Joonis 7. Otsingu eeltöötluses piksli *p* vastete otsimine.

Kuna kõrgeim tase on väga väike, siis toimub jõumeetodil otsing kiiresti. Liikudes püramiidis allapoole kasutatakse igal järgneval tasemel eelmisel tasemel leitud vasteid, millega välditakse paljude ümbruste võrdlemist ja kulutatakse seega vähem aega. Nagu k-koherentsusega otsing sünteesi ajal, ei leia ka MRPM alati tegelikke parimaid vasteid, aga ajaline võit on piisavalt suur, et õigustada natuke halvemat kvaliteeti.

Püramiidi kõrgematel tasemetel on vähem informatsiooni esialgse pildi kohta, mistõttu võib juhtuda, et madalamatele tasemetele jõudes on mõned tegelikult head vasted kõrgematel tasemetel kõrvale jäetud. Probleemi vältimiseks on MRPM autoritel kaks lahendust. Neist esimest illustreerib joonis 8. Võrreldakse pikslite *p* ja *q* ümbrusi tasemel $t_1$. Olgu $(x, y)$ *p* koordinaadid. Tasemel $t_2$ vastab pikslile *p* piksel *p'* koordinaatidega $(2x, 2y)$. Samamoodi on leitud piksli *q* järgi *q'*. Tähistagu $\vec{x}$ piksli *x* ümbruse vektorit. *p* ja *q* ümbruste erinevus on $\|\vec{p}-\vec{q}\|^2+\|\vec{p'}-\vec{q'}\|^2$. Idee on selles, et vähendamise käigus kaotsi läinud informatsioon on madalamal tasemel veel alles. Töös loodud lahenduses võrreldakse kolme, mitte kahe järjestikuse taseme ümbruseid.




Joonis 8. Täiendatud ümbruste võrdlemine otsingu eeltöötluses.

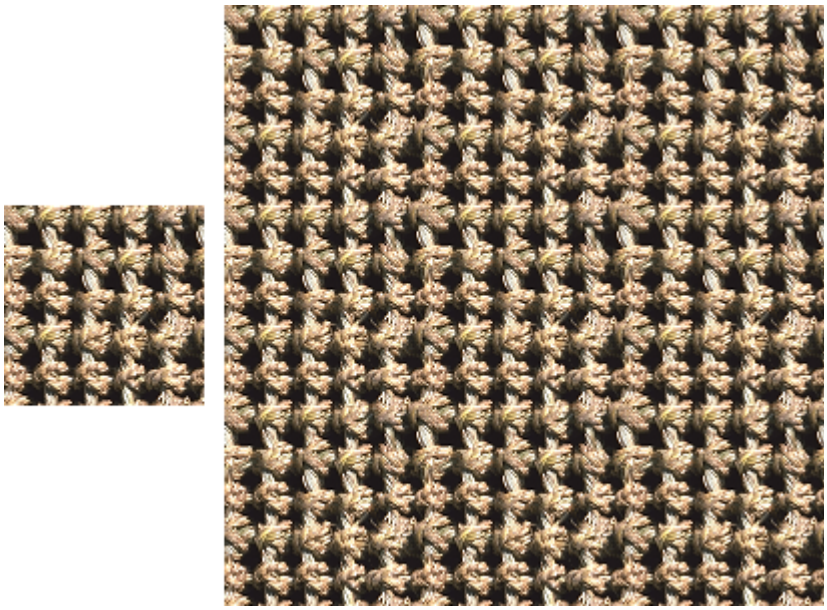
Teine vahend halbade otsuste vältimiseks on kõrgematel tasemetel suurema *k* väärtuse kasutamine. Ainult kõige madalamal tasemel otsitakse *k*-1 vastet. Liikudes püramiidis alt üles suureneb käesoleva töö implementatsioonis iga tasemega *k* väärtus 3 võrra, kuni ei ületata 30-t. Kuigi sünteesi algoritm kasutab leitud vastetest jätkuvalt vaid *k*-1 parimat vastet, vähendab see kõrgematel tasemetel kõrvale jäetud heade vastete arvu.

Sünteesimisel kasutatakse püramiidi, kus mingil tasemel sünteesides on vaja vasteid lähtepildi sellest tasemest. Seetõttu ei saa eeltöötlust täielikult läbida enne sünteesimist. Igal tasemel tuleb enne optimeerimist leida MRPM algoritmiga eelmise taseme vastetest optimeeritava taseme vasted.

## 2.5. Servade käsitlemine

Pildi servas asuva piksli ümbrus ei pruugi pildi sisse ära mahtuda ehk ümbrus ulatub üle ääre. Käesolevas töös välditakse selliseid lähtepildi ümbrusi. Sünteesitavas tekstuuris seda aga teha ei saa. Artiklis [13] soovitatakse pildi kõrvale jäävat ala kujutada pildi peegeldusena ehk pildist vasakul ja paremal on pildi peegeldus *y*-telje suhtes, üleval ja all peegeldus *x*-telje suhtes, nurkades peegeldus *x*- ja *y*-telje suhtes.

Artiklis [13] on kirjeldatud veel teist lihtsat võimalust. Lugedes/kirjutades pikslit koordinaatidega $(x, y)$ pildilt mõõtmetega *l*-laius, *k*-kõrgus, loetakse/kirjutatakse piksel koordinaatidega $(x \bmod l, y \bmod k)$. See annab algoritmile omaduse, mida esialgses tekstuurisünteesi definitsioonis pole, nii öelda servadeta tekstuuride sünteesi. Asetades kõrvuti koopiaid sünteesitud pildist, pole koopiate vahelisi servi märgata. Seda sellepärast, et vastasservade pikslid mõjutavad üksteist sünteesimise käigus. Omadus on kasulik 3D graafikas. Kui kunstnikul on vaja katta mingi suur pind tekstuuriga, siis mälu säästmiseks teeb kunstnik sellise tekstuuri, mille koopiaid saab geomeetrilisele pinnale märkamatult kõrvuti asetada. Seda illustreerib joonis 9. Käesolevas töös loodud lahenduses on kasutajale antud võimalus mõlema meetodi vahel valida.



Joonis 9. Vasakul on töös loodud lahenduse sünteesitud nii-öelda servadeta tekstuur. Paremal on sellest üheksa kõrvuti asetatud koopiat. Tekstuuri lähtepilt pärineb allikast [14].

# 3. Tulemused

## 3.1. Implementatsioon

Algoritm on programmeeritud keeles C, sest seda keelt kasutavad nii GIMP kui GEGL.

Algoritm töötab CIE Lab (edaspidi Lab) värviruumis, millele on lisatud läbipaistvuse komponent. Lab on rahvusvahelise komisjoni CIE (prantsuse keeles *Commission internationale de l'éclairage*) loodud standardiseeritud värviruum. Seda värviruumi toetab babl, mis võimaldab värviruumi Lab + läbipaistvus viia kõiki pilte, mida GEGL avada suudab. Lab loodi värvide erinevuse võrdlemiseks. Kahe piksli vektorite vaheline kaugus on kooskõlas inimese nägemistajuga. Kuigi efektiivsuse eesmärgil kasutatakse töös vektorite kauguse ruutu, andis ikkagi Lab värviruumis töötamine paremaid tulemusi kui babli RGB värviruum (mida GIMP kasutab kõige sagedamini).

Värvide esitamiseks kasutatakse 32-bitiseid ujukomaarve. Katsetused erinevate arvutüüpidega on näidanud, et mida suurem arvutüüp, seda aeglasemalt pilditöötlusoperatsioonid töötavad. Selle ja ilmselt vähese optimeerituse tõttu töötab programm suhteliselt aeglaselt. Kuna kvaliteet on oluline, siis ei saa lubada 8-bitiste arvude kasutamist. Kiiruse tõstmiseks on kasutatud mitmeid võtteid.

Algoritm töötab paralleelselt mitme lõimena. Selleks on kasutatud Fortrani, C ja C++ kompilaatorite direktiivide kogumikku ja rakendusliidest OpenMP, mis lihtsustab paralleelprogrammeerimist. GEGL ja GIMP kompileeritakse GCC (*GNU Compiler Collection*) kompilaatoriga, millel on OpenMP tugi. Kuna töös on kasutatud ainult kompilaatori direktiive, mitte rakendusliidest, siis OpenMP toeta kompilaator tekitab ühelõimelise programmi. Paralleelselt täidetakse neli osa algoritmist:

- otsingu eeltöötlus;
- M-samm;
- E-samm;
- pildi suurendamine/vähendamine.

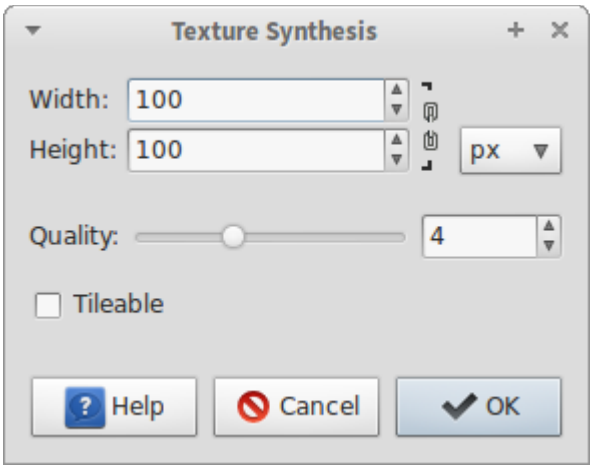
Iga lõim töötleb mingi hulga ridu pildist. Algoritmide olemuse tõttu on kõikide ridade

töötlemine iseseisev ja jagatud andmeid ei kirjutata. Seetõttu pole vaja lukustamist ning algoritmi paralleelsena realiseerimine on kerge ja efektiivne.

Otsides mingile ümbrusele vastet M-sammus või otsingu eeltöötluses jätab programm meelde seni parima vaste erinevuse. See võimaldab erinevuste arvutamise pooleli jätta, kui erinevus ületab seni väikseima erinevuse.

Salvestades optimeerimise vahetulemusi oli märgata, et tekstuurid, mille sünteesimisel läbiti maksimaalne lubatud arv iteratsioone, muutusid viimaste iteratsioonide käigus nii vähe, et muutused pole tajutavad. Esialgses algoritmis katkestati optimeerimine iteratsioonide piirini jõudes või kui kahe järjestikuse M-sammu väljund oli sama. Käesolevas töös loetakse kokku M-sammu käigus eelmise M-sammuga võrreldes muutunud vastete arv. Itereerimine katkestatakse kui muutunud ja muutumata vastete suhe ületab 0,002. Maksimaalne lubatud iteratsioonide arv on 60.

Algoritmi GIMPis kasutamiseks loodi GIMPi pistikprogramm, mis kasutab algoritmi läbi GEGLi. GIMP kasutab kasutajaliideste teeki GTK+ (*GIMP Toolkit*) ja seetõttu on ka pistikprogrammi dialoog programmeeritud GTK+ abil. Joonisel 10 on toodud ekraanitõmmis dialoogist. Parameeter *width* ja *height* on laius ja kõrgus, *quality* on vahemikus 1 kuni 10 ja selle väärtusest ühe võrra suuremat arvu kasutatakse parameetri *k* väärtusena, *tileable* sünteesib nii-öelda servadeta tekstuuri.




Joonis 10. Ekraanitõmmis GIMPi pistikprogrammi dialoogist.

## 3.2. Võrdlus teiste lahendustega

GIMPile on programmeeritud mitu erinevate lisavõimalustega tekstuurisünteesi pistikprogrammi. Peatükis antakse nendest ülevaade ning võrreldakse nendega töö käigus loodud lahendust.

Cornet ja Rouquier pistikprogramm [15] kasutab *patch-based* algoritmi. See on küll heade tulemustega ja kiire, aga toetab ainult tavalist tekstuurisünteesi ja nii-öelda servadeta tekstuuride sünteesi (asetades kõrvuti tekstuuri koopiaid, pole koopiate vahelisi servi märata). *Pixel-based* algoritmid (ka käesolevas töös kasutatud algoritm) on paindlikumad ning võimaldavad enam kasulikke ülesandeid täita.

Harrison [4] on enda loodud algoritmi realiseerinud GIMPi pistikprogrammina nimega Resynthesizer. Tegu on kiire *pixel-based* algoritmiga, mis toetab lisaks tavalisele tekstuurisünteesile nii-öelda servadeta sünteesi, *image completion*'it (pildilt automaatselt objektide eemaldamine) ja tekstuuri ülekannet ühelt pildilt teisele (teise pildi objektide struktuur säilib, aga pinnal on tekstuur esimeselt pildilt). Tegemist on väga populaarse pistikprogrammiga, mille enimkasutatud võimalus on *image completion*. Pistikprogrammi peamised eelised käesolevas töös loodud meetodi ees on kiirus ja *image completion*'i tugi. Puuduseks on vana algoritm, mis annab häid tulemusi vaid lähtepiltidega, millel pole struktuuri. Näiteks muru tekstuuriga annab algoritm häid tulemusi, aga kiviseinaga mitte.

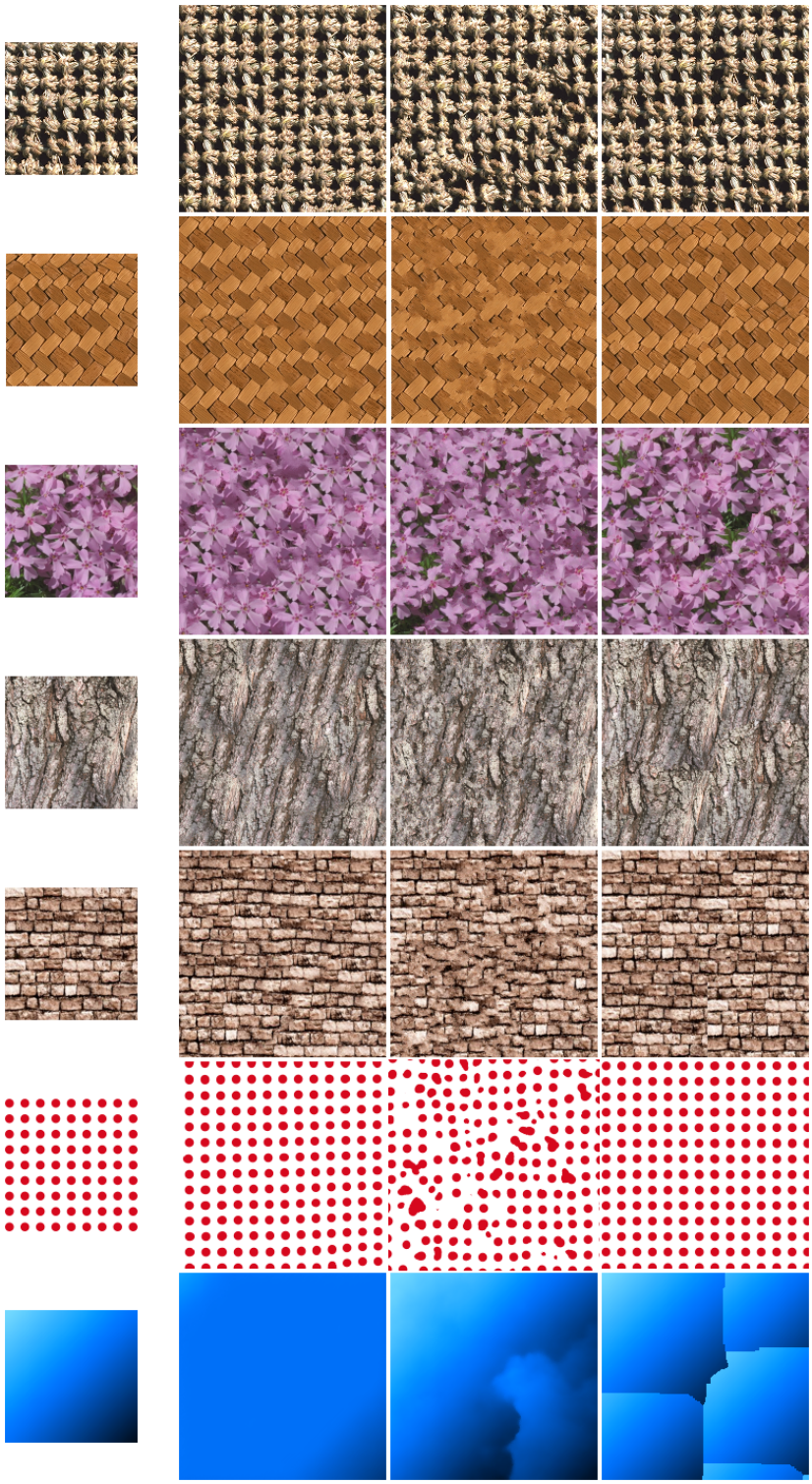
Sama autori loodud on teinegi pistikprogramm nimega TextureOps. See kasutab väga lihtsat meetodit, mis kleebib kokku tükke sisendpildist. Kuigi see meetod on väga kiire, annab see häid tulemusi väheste piltide korral. Programm toetab servadeta tekstuuride sünteesimist ja mõningaid lisavõimalusi, mis pole seotud tekstuurisünteesiga. Eelis on kiirus ja puudus on halvad tulemused.

Irwin [16] kasutab *pixel-based* meetodit *image completion*'i jaoks. Pistikprogramm ei toeta tavalist tekstuurisünteesi ning algoritm on väga aeglane (artiklis on toodud näited, kus sünteesimisele kulus mitukümmend minutit). Programmi ega selle lähtekoodi veebist ei leidunud.

Lisaks kirjeldatutele, on neil kõigil veel mitu puudust.

1. GIMPi versioon 2.10 toetab vanade pistikprogrammide kasutamist, aga alates versioonist 3.0 tagasiühilduvus kaotatakse ja olemasolevaid pistikprogramme enam kasutada ei saa.
2. Kuigi versioonis 2.10 saab vanu pistikprogramme kasutada, siis töötavad need jätkuvalt 8-bitiste RGBA piltidega. Kui kasutaja töötleb näiteks 32-bitist RGBA pilti, siis vanade pistikprogrammide kasutamiseks teisendatakse pilt vanasse esitusse ja pärast jälle tagasi, mille tagajärjeks on informatsiooni kadu.
3. Kuna pistikprogrammid on programmeeritud GIMPi liidestega, siis ei saa neid teistes programmides kasutada.

Joonisel 11 on võrreldud Resynthesizeri, Texturize'i ja käesolevas töös loodud lahendust erinevate tekstuuride sünteesimisel. Töös realiseeritud algoritmi halva jõudluse näiteks kulub joonisel 11 oleva kiviseina sünteesimiseks autori arvutil umbes 23 sekundit, Resynthesizeril umbes sekund ja Texturize'il alla sekundi.



Joonis 11. Võrdlus vanemate pistikprogrammidega. Esimeses veerus on lähtepilt, teises veerus töös realiseeritud lahenduse väljund (parameetri k väärtus oli 5), kolmandas Resynthesizeri väljund, neljandas Texturize'i väljund.  Esimesed neli lähtepilti pärinevad allikast [14].

# 4. Täiustamisvõimalused

Tekstuurisünteesi üheks populaarsemaks kasutusalaks on *image completion*, kus kasutaja annab ette pildi, märgistab pildil mingi ala ja algoritm katab selle ülejäänud pildi põhjal. Seda kasutatakse piltidelt mittesoovitavate kujutiste eemaldamiseks. See on rastergraafikas sagedasti esinev ülesanne, mis nõuab käsitsi teostamisel oskusi ja palju aega.

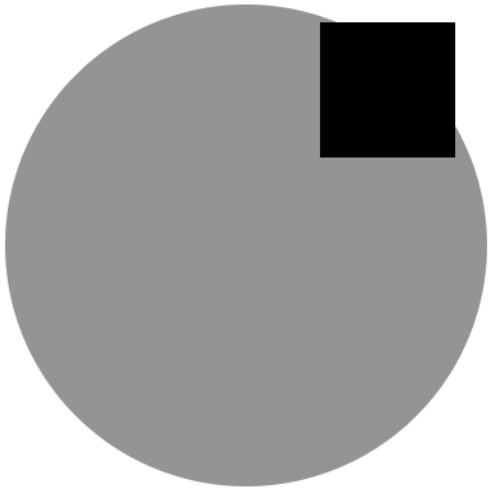
Kuigi M-sammu otsinguks kasutatud algoritmid k-koherentsus ja MRPM parandavad kõvasti töö kiirust jõumeetodil otsinguga võrreldes, kulub siiski kõige enam aega M-sammus. Artikli [10] algoritmis läbitakse lisaks k-koherentsuse eeltöötlusele veel peakomponentide analüüsi. See on statistiline meetod, mis võimaldab suure muutujate arvuga andmete korral leida väiksem hulk muutujaid, mis andmeid kirjeldaksid. Tekstuurisünteesi seisukohalt võimaldab see ümbruste võrdlemisel vähem arvutusi teha. Näiteks on mõistlik ümbruse suurus $9\times9$, mille korral arvutatakse vektorite kauguse ruut 243-dimensioonilises ruumis (kui töötatakse RGB värviruumis). Kasutades peakomponentide analüüsi võrreldakse [10] meetodis vaid 16-dimensioonilisi vektoreid. Nagu teiste otsingu kiirendamise võimalustega kaasneb ka sellega natuke halvem kvaliteet.

Algoritm ei kasuta keerulisi andmestruktuure ning seda on kerge paralleelsena programmeerida. Need kaks asjaolu teevad võimalikuks programmeerida algoritm graafikakaardil. Graafikakaardi protsessoril on palju tuumasid, mis võimaldab efektiivselt täita ülesandeid, kus on võimalik töö tuumade vahel jagada. Artikkel [10] kirjeldab k-koherentsuse ja tekstuuri optimeerimisega algoritmi realiseerimist graafikakaardil. Ka MRPM algoritm on implementeeritud graafikakaardil [12]. GEGLis on programmeerimiskeele OpenCL tugi, mille kompilaatorid (tavaliselt on kompilaator graafikakaardi draiveriga kaasas) võimaldavad OpenCL-is kirjutatud programme graafikakaardile kompileerida.

Viimastel aastatel on loodud mitmeid algoritme ümbruste vastete kiiresti otsimiseks. Ka töös kasutatud MRPM on loodud selle, mitte ilmtingimata k-koherentsuse eeltöötluse tarbeks. On võimalik, et nendest algoritmidest mõne kasutamine k-koherentsuse ja MRPM-i kombinatsiooni asemel annaks kiirema M-sammu. Mitmed vastete otsimise algoritmid

kasutavad aga keerulisi andmestruktuure (erinevad puud), mis võivad graafikakaardil programmeerimise keeruliseks teha.

*Image completion*'i korral kasutatakse algoritmi piltidel, mis ei pruugi olla tekstuurid. Sellistes piltides ümbruseid otsides on tihtipeale head vasted mõnes mõttes valel kujul. Mitmedes töödes (näiteks [17, 18]) loodud algoritmid kasutavad ümbruseid otsides ka ümbruste kandidaatide muutmist, näiteks pööramist, hele-tumeduse, kontrasti või suuruse muutmist. See võimaldab palju usutavamaid tulemusi. Probleemi illustreerib joonis 12. Kasutaja soovib täita musta ala. Tulemus peaks olema ring. Pildis ei leidu ümbrusi, mis sobiksid selle ala täitmiseks, aga ümbrusi pöörates on võimalik ala täita, nii et tulemuseks oleks ring.



Joonis 12. Näide pöörete vajalikkusest *image completion*'il.

# 5. Kokkuvõte

Tekstuurid on mustrid, mis 3D graafikas kantakse detailide lisamiseks geomeetrilise objekti pinnale. Tekstuurisünteesi eesmärk on luua lähtepildi põhjal suvaliste mõõtmetega tekstuure. Tekstuurisüntees on leidnud palju kasutust ka mujal kui 3D graafikas. Töö käigus realiseeriti tekstuurisünteesi algoritm rastergraafikaredaktorile GIMP. Lisaks tavapärasele tekstuurisünteesile toetab lahendus nii-öelda servadeta tekstuuride sünteesimist (asetades sünteesitud tekstuurist mitu koopiat kõrvuti pole koopiate vahelisi servi märgata).

Algoritm, mis baseerub artiklil [7], alustab juhuslikult valitud pikslite kopeerimisega sisendist väljundisse ning seejärel optimeerib väljundit, et muuta seda lähtepildile sarnasemaks. Algoritmi peamine puudus on kiirus, mis on tingitud vajadusest otsida lähtepildist alasid, mis sarnaneksid mingi alaga väljundis. Otsingu kiirendamiseks kasutati püramiide ning algoritme k-koherentsus [11] ja *mixed-resolution patch-matching* [12].

Algoritm realiseeriti pilditöötluse teegi GEGL osana programmeerimiskeeles C. GIMPile loodi pistikprogramm, mis võimaldab algoritmi läbi GEGLi kasutada. Töö kiirendamiseks kasutati paralleelprogrammeerimist.

Realisatsiooni on võrreldud varasemate GIMPile loodud tekstuurisünteesi pistikprogrammidega. Tulemused on head, aga sünteesimine võtab kaua aega.

Praktiliseks kasutamiseks tuleks algoritm kiiremaks teha. Selleks on töös välja toodud erinevaid täiustamisvõimalusi. Tekstuurisünteesi üks populaarsemaid rakendusi on *image completion,* mis võimaldab kasutajatel automaatselt piltidelt objekte eemaldada. Algoritm on paindlik ning oleks võimalik ja kasulik sellele *image completion* lisada.

Töö ilmumist on toetanud IT akadeemia.

# Kirjandus


[1] L.-Y. Wei, S. Lefebvre, V. Kwatra, ja G. Turk, „State of the Art in Example-based Texture Synthesis“, *Eurographics 2009, State of the Art Report, EG-STAR*, 2009.

[2] R. C. Gonzalez ja R. E. Woods, *Digital Image Processing*, 2 tr. Upper Saddle River, New Jersey, USA: Prentice Hall, 2002.

[3] Tundmatu autor, „Red parrot“, *Flickr*. http://www.flickr.com/photos/doug88888/5910019136/ [28. aprill 2013].

[4] P. F. Harrison, „Image Texture Tools: Texture Synthesis, Texture Transfer, and Plausible Restoration“, PhD, Monash University, 2005.

[5] A. A. Efros ja T. K. Leung, „Texture Synthesis by Non-Parametric Sampling“, *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, Washington, DC, USA, 1999, lk 1033–.

[6] A. A. Efros ja W. T. Freeman, „Image quilting for texture synthesis and transfer“, *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, New York, NY, USA, 2001, lk 341–346.

[7] V. Kwatra, I. Essa, A. Bobick, ja N. Kwatra, „Texture optimization for example-based synthesis“, *ACM Trans. Graph.*, kd 24, nr 3, lk 795–802, juuli 2005.

[8] F. Neyret ja M.-P. Cani, „Pattern-based texturing revisited“, *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, New York, NY, USA, 1999, lk 235–242.

[9] R. Fisher, S. Perkins, A. Walker, ja E. Wolfart, „Spatial Filters - Gaussian Smoothing“, *Hypermedia Image Processing Reference*. http://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm [28. aprill 2013].

[10] H.-D. Huang, X. Tong, ja W.-C. Wang, „Accelerated parallel texture optimization“, *J. Comput. Sci. Technol.*, kd 22, nr 5, lk 761–769, september 2007.

[11] X. Tong, J. Zhang, L. Liu, X. Wang, B. Guo, ja H.-Y. Shum, „Synthesis of bidirectional texture functions on arbitrary surfaces“, *ACM Trans. Graph.*, kd 21, nr 3, lk 665–672, juuli 2002.

[12] H. Sureka ja P. J. Narayanan, „Mixed-resolution patch-matching“, *Proceedings of the 12th European conference on Computer Vision - Volume Part VI*, Berlin, Heidelberg, 2012, lk 187–198.

[13] L.-Y. Wei ja M. Levoy, „Fast texture synthesis using tree-structured vector quantization“, *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, New York, NY, USA, 2000, lk 479–488.

[14] Massachusetts Institute of Technology, „Vision Texture“, *MIT Media Lab: VisMod Group*. http://vismod.media.mit.edu/vismod/imagery/VisionTexture/ [28. aprill 2013].

[15] E. Cornet ja J.-B. Rouquier, „The Texturize plugin for The GIMP“. http://gimp-texturize.sourceforge.net/ [8. aprill 2013].

[16] C. Irwin, „The Development of a Fragment-Based Image Completion Plug-in for the GIMP“, PhD, Rhodes University, Grahamstown, South Africa, 2004.

[17] S. Darabi, E. Shechtman, C. Barnes, D. B. Goldman, ja P. Sen, „Image melding: combining inconsistent images using patch-based synthesis“, *ACM Trans. Graph.*, kd 31, nr 4, lk 82:1–82:10, juuli 2012.

[18] A. Mansfield, M. Prasad, C. Rother, T. Sharp, P. Kohli, ja L. Van Gool, „Transforming Image Completion“, *British Machine Vision Conference (BMVC)*, 2011.

# Summary

**Texture Synthesis for the Raster Graphics Editor GIMP**

**Bachelor's thesis**

**Ville Sokk**

In 3D graphics, textures are patterns that are mapped on to geometrical objects to add detail. The goal of texture synthesis is to generate textures of arbitrary size based on an example image. In this thesis a texture synthesis algorithm was implemented for the raster graphics editor GIMP. In addition to regular texture synthesis, synthesis of tileable textures (if multiple copies of the texture are placed side by side then the edges between copies are not noticeable) was implemented.

The thesis consists of an overview of texture synthesis research, description of the used algorithm, description of the implementation, comparison to existing GIMP texture synthesis plugins and recommendations for further improving the algorithm and the implementation.

The algorithm, which is based on article [7], generates an estimation of the texture and then improves it using iterative optimization. The main issue with the algorithm was speed. Multiple algorithmic improvements and parallel programming were used to increase performance. Compared to existing plug-ins, the implementation achieves good results but takes a lot of time.

Recommendations for further work were given to improve performance and quality of both the algorithm and implementation. A useful additional feature would be image completion which allows automatic removal of objects from images.

This work was supported by the IT Academy program.

# Lisad

Lähtekood on saadaval aadressil:
https://drive.google.com/folderview?id=0B216_JTMLR6_Q0RTdllNa0lCNE0&usp=sharing

Arhiivis *kood.tar.gz* on GEGLile lisatud kood failis *gegl.c* ja GIMPile lisatud kood failis *gimp.c*.

Arhiivides *gegl-0.2.1.tar.bz2* ja *gimp-2.9.1.tar.bz2* on vastavalt GEGLi ja GIMPi lähtekood koos töö käigus tehtud muudatustega. GEGLile lisatud kood on failis *operations/workshop/texture-synthesis.c* ja GIMPile lisatud kood failis *plug-ins/common/texture-synthesis.c*. GIMP ja GEGL on saadud arendusjärgus versioonidest ja võivad olla ebastabiilsed ning halva jõudlusega. GEGLi kompileerimisel on *configure* skripti käivitamisel vajalik lipp *--enable-workshop*. OpenMP toe saab lipuga *CFLAGS="-fopenmp"*.

GIMPis saab pistikprogrammi dialoogi avada menüüst *Filters* ➔ *Map* ➔ *Texture Synthesis*.

**Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks**

Mina Ville Sokk
(sünnikuupäev: 29. märts 1991)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Tekstuurisüntees rastergraafikaredaktorile GIMP“ mille juhendaja on Sven Laur,

1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, 8. mai 2013