SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 13

Postupci sinteze tekstura i teksturiranje objekata

Denis Tošić

Zagreb, lipanj 2008

Posebna zahvala mojoj profesorici i mentorici
Željki Mihajlović na pruženoj podršci i pomoći,
gospodinu Johannesu Kopfu
za male savjete od velikog značenja,
kolegi Vjekoslavu Vuliću na satima
provedenima uz kavu i konstruktivne rasprave,
te mojoj obitelji i prijateljima.

Uvod

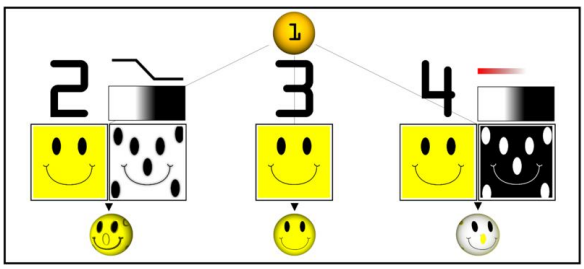
Teksture su jedan izuzetan i sveprisutan element računalne grafike. Pomoću njih mnoga svojstva i pojave možemo učiniti izuzetno realnima, npr. izgled (površinske boje nekog tijela), gibanje (kretanje životinja, valovi tekućina) itd. Kako je glavni zadatak računalne grafike reproduciranje stvarnog svijeta na računalnom ili nekom drugom zaslonu, generiranje i primjena tekstura, teksturiranje, predstavljaju vrlo važan čimbenik tog znanstvenog područja.

Ovaj rad prikazuje rezultate istraživanja tekstura i njihovog sintetiziranja. Cilj je bio razviti učinkoviti algoritam te njegovu implementaciju koja će za neki dati uzorak generirati teksturu proizvoljne veličine. Jedini zahtjev koji je bio postavljen jest da generirana tekstura bude slična danom uzroku. Pri tome riječ „slična“ znači da promatrač kojem su dani uzorak i sintetizirana tekstura može pretpostaviti da su oni dio iste cjeline, ili da su nastale na isti ili sličan način.

1. Teksture i sinteza tekstura

Teksture predstavljaju vidljive ili opipljive površine koje su izgrađene na temelju jednog ili više neprestano ponavljajućih uzoraka (npr. niti mreže).

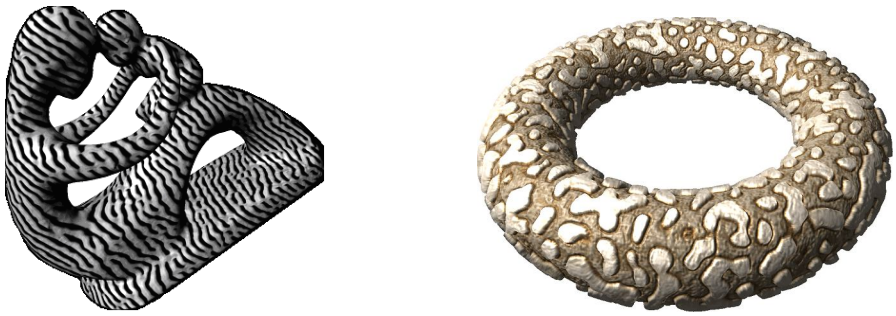
Teksturiranje jest postupak kojim se objektu dodavaju razna svojstva, poput boje, prozirnosti, hrapavosti, osjevjetljena itd., u vidu povećanja realističnosti njegova prikaza. Postoje dva temeljna načina teksturiranja: prvi je preslikavanje tekstura (*eng. texture mapping*), koji se odnosi na teksturiranje površine objekta, odnosno njezinih poligona. Najjednostavnije rečeno, ovaj postupak jest poput lijepljena uzorkovanog papira na stranice jedne bijele kocke (slika 1.). Postupak preslikavanja počinje se primjenjivati od sedamdesetih godina prošlog stoljeća, a utemeljio ga je trenutni predsjednik Disneyevih i Pixarovih animacijskih studija, dr. Edwin Catmull (1945. -). Dr. Catmull je, pored postupka mapiranja, zaslužan i za mnoštvo drugih važnih izuma u računalnoj grafici, primjerice z-spremnika (*eng. z-buffer*), te algoritama koji se primjenjuju u postupku antialiasinga.



Slika 1.1 Jednostavni primjer mapiranja tekstura

1. neteksturirana sfera, 2. 3. i 4. mapirane teksture

Drugi pristup jest u potpunosti drugačiji, a naziva se hiperteksturiranje ili izgradnja čvrstih tekstura. Umjesto teksturiranja same površine objekta „lijepljenjem“ 2D uzoraka na poligone površine tijela, ovim se postupkom oblikuje i unutrašnjost objekta. Naime, korištenjem hipertekstura, za razliku od običnih dvodimenzijalnih tekstura, definiramo i svojstva unutrašnjosti objekta. Naravno, i to ima svoje prednosti i nedostatke.



Slika 1.2 Objekti modelirani volumnim teksturama

Glavna prednost ovog postupka jest što su objekti koji su izgrđeni na ovaj način vrlo detaljno opisani, oni sadrže velike količine podataka koji se mogu izravno koristiti u izračunima, primjerice, njihove deformacije (pucanje kamena na dva dijela). U takvim slučajevima nije potrebno teksturiranje novih površine, jer su nam njihove vrijednosti već poznate od prije. Loša strana ovog postupka jest što takvi elementi zauzimaju golemu količinu memorijskog prostora.

Međutim, glavni nedostatak hipertekstura jest vrijeme potrebno za njihovo generiranje. Kako se radi o velikoj količini podataka, i velikom broju računskih operacije, za generiranje malih, pravilnih objekata (primjerice kocke), potrebna je, u okviru brzine današnjih računalnih procesora, ogromna količina vremena. Na sreću, kada jednom teksturiramo objekt hipertekstorom, mi ga uvijek možemo iznova koristiti, bez potrebe za ponovnim postupkom sinteze.

2. Boja i slikovni formati

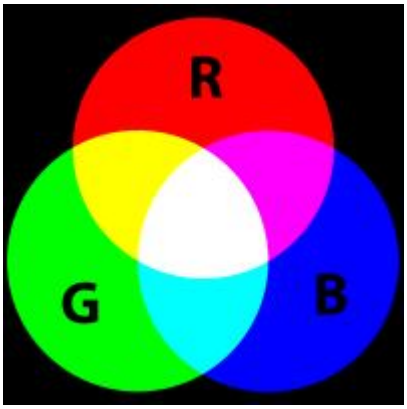
Prije nego što dublje zaronimo u područje dvodimenzionalnih i trodimenzionalnih tekstura, potrebno je razmotriti svojstva objekata čijim procesuiranjem te strukture nastaju. Drugim riječima, našu ćemo pažnju na trenutak posvetiti bojama i slikovnim formatima.

2.1 Modeli boja, RGB

Model boja jest apstraktni matematički model koji opisuje kako se različite boje mogu zapisati kao nizovi brojeva. Zavisno o modelu, tih nizova (komponenti) može biti od jedan do četiri, s time da što ih je više, te što je veličina tih nizova veća, to je raspon boja koji se može prikazati modelom veći. Svaki od tih nizova predstavlja jednu osnovnu boju, te se njihovom kombinacijom generiraju sve ostale.

Postoje dva različita temeljna modela boja: aditivni i subtraktivni. Razlika između njih jest način na koji se generira boja iz komponenti. Kod aditivnog modela boja se dobiva zbrajanjem pojedinih komponenti (npr. RGB model), a kod subrativnog oduzimanjem (npr. CMYK model). Oba modela ostvaruju golemi raspon boja, no kako se CMYK uglavnom primjenjuje u tiskarskoj industriji, mi ćemo našu pažnju usmjeriti ka RGB modelu.

Akronim RGB predstavlja riječi *Red* (crvena), *Green* (zelena) i *Blue* (plava). Crvena, zelena i plava su, naime, temeljne, tj. osnovne komponente ovog modela. Svaka pojedina komponenta zapisuje se u niz od 8 bita, što znači da svaka boja ima sveukupno 256 različitih nijansi, što se na prvi pogled može činiti malo. Međutim, kako postoje tri različite komponente, RGB model može ukupno prikazati 256^3 različitih boja, što je više nego što ljudsko oko može raspoznati.



Slika 2.1.1 Prikaz RGB modela boja

Glavna primjena ovog modela jest u prikazivanju slika na računalnom i televizijskom ekranu, a koristi (u manjoj mjeri) i u fotografiji. Treba međutim napomenuti da se iste boje kodirane RGB modelom ne moraju nužnu prikazivati na isti način različitim uređajima. Naime, svaki uređaj ima svoj vlastiti sustav upravljanja prikazom i bojama.

Kako je RGB aditivni model, boje se dobivaju zbrajanjem pojedinih komponenti. Tako se recimo trojka vrijednosti (255, 255, 0) preslikava u žutu boju najjačeg intenziteta, (255,255,255) se preslikava u bijelu, dok se trojka (0,0,0) preslikava u crnu.

RGB model se koristi kako kod ulaznih tako i kod izlaznih uređaja. U ulazne spadaju npr. TV i video kamere, skeneri i digitalne kamere, dok u izlazne spadaju LCD i CRT ekrani, plazma uređaju, ekrani mobilnih telefona itd.

2.2 BMP slikovni format

BMP datotečni format, ponekad zvan još i bitmap, jest slikovni format gdje su bitovi podataka organizirani u dvodimezionalnom polju , tj. mapi (otuda i naziv bitmap – mapirani bitovi). Kvaliteta slike zapisane u BMP formate ponajviše zavisi o dubini boje (*eng. color depth*), odnosno broju bitova koji određuju vrijednost jednoga piksela. Taj broj poprima jednu od slijedećih vrijednosti : 1, 4, 8, 16, 24 ili 32, pri čemu su danas najčešće 24 ili 32.

Za vrijednosti 1 slika jest crno-bijela, za 4 sadrži i nijanse sive, dok je za ostale vrijednosti slika u boju. Treba doduše napomenuti da je dubinom boje (*eng. color depth*) iznosa 8 bita također moguće prikazati slike u crno-bijeloj varijanti s nijansama sive. Za dubinu boje iznosa 24 BMP koristi klasični RGB model, dok za dubinu 32 koristi RGB model i još jednu dodatnu komponentu – prozirnost.

Podatci u BMP formatu podijeljeni su u četiri bloka:

- BMP zaglavlje (*eng. header*) – općeniti podatci o datoteci (npr. veličina i vrsta)
- Bitmap podatci – detaljniji opis kompletne datoteke
- Tablica boja – popis boja ako se koristi indeksirani pristup opisivanja vrijednosti piksela
- Bitmap podatci – vrijednosti svakog pojedinog piksela

Svakako najvažniji sadržaj ovog formata jesu vrijednosti piksela. Indeksirani pristup koristi se kada je raspon boja mali, odnosno ako je dubina boje 1, 4, ili 8. Time se definira raspon indeksa do najviše 256, što je upravo i najveći raspon boja u ovakvom pristupu. U tablici jest svakome indeksu pridodana različita boja, dok se u bloku podataka za svaki piksel navodi samo indeks boje koju on sadrži. Ako je dubina boje 16 ili veći, tada u BMP formatu ne postoji tablica boja jer vrijeme iscrtavanja na zaslon drastično raste zbog neprestanog pristupa tablici boja, te zbog velike količine memorije koju ona zauzima. Vrijednost boje uvijek je zapisana u RGB formatu, koji će u nastavku biti detaljnije razmotren.

U slučaju kada je dubina boje 16 ili veći, u datoteci su izravno zapisane vrijednosti za svaki piksel u RGB formatu. Vrijednosti se nižu počevši iz donjeg lijevog ugla, te se zatim opisuje redak po redak, sve do zadnjeg piksela koji se nalazi u gornjem desnom uglu.

3. Sinteza tekstura

Kao što je već jednom rečeno, razlikujemo dvije vrste tekstura: dvodimenzionalne ili plošne, i trodimenzionalne, prostorne ili hipertekstura. Postoji više različitih pristupa za njihovo generiranje, koji se više-manje temelje na sličnim načelima. Međutim, treba napomenuti da iako su ta načela vrlo slična, konačna implementacija i svojstva tih tekstura se uvelike razlikuju.

3.1 Obilježja i sinteza 2D tekstura

Dvodimenzionalne, odnosno plošne teksture već su dugi niz godina aktivno područje istraživanja, i to ne samo računalne grafike, već i drugih znanstvenih grana, poput računalnog vida ili obrade slike. Postoji mnoštvo pristupa ovom problemu, te stoga ovdje navodimo nekoliko najutjecajniji.

3.1.1 Fizikalna simulacija

Fizikalna simulacija predstavlja jedan vrlo rano rješenje problema sinteze. Naime, tekstura se generira tako da se simulira prirodni proces izgradnje tvari čija se tekstura želi sintetizirati. Na taj se način mogu dobiti vrlo realistične teksture poput kože, krzna ili ljuske. Međutim, kako se procesi izgradnje pojedinih tvari uvelike razlikuju, ovaj se model sinteze učinkovito može primijeniti na vrlo uzak raspon tekstura.

3.1.2 Markovljeva slučajna polja

Markovljeva slučajna polja kao matematički model zavisnih slučajnih varijabli također su našla primjernu u problemu sinteze tekstura. Na mnoštvu raznih primjera pokazano je da je ova metoda vrlo učinkovita i da daje realistične rezultate. Međutim, i ova se metoda vrlo rijetko primjenjuje zbog izuzetno velikih zahtjeva koje postavlja računalu. Naime, i za sintezu vrlo malih tekstura potrebna

je ogromna količina vremena; dulja trajanja procesa sinteze može se kretati u rasponu od nekoliko sati pa do nekoliko dana!

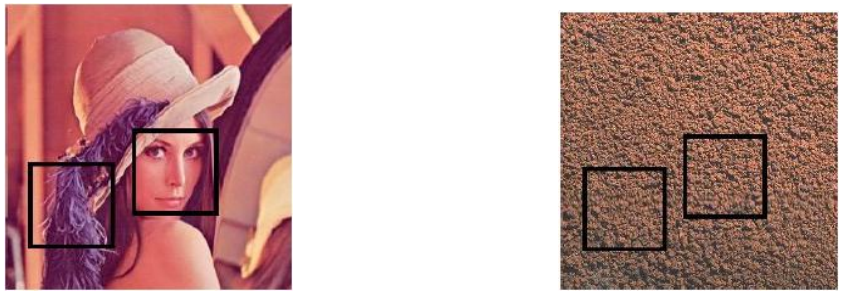
3.1.3 Usklađivanje značajki (*feature matching*)

Metoda usklađivanje značajki (općenito izbjegavajte zamjenice) najjače je zastupljena u okviru problematike sinteze teksture. Nove se teksture generiraju „slaganjem“ komponenti uzorka u veće cjeline. Prilikom odabire komponente promatra se međusobna sličnost trenutno promatranog prozora teksture kojeg prilagođavamo i uzroka. Treba napomenuti da riječ sličnost može imati više različitih interpretacija: od vektorske udaljenosti pa sve do razlike u statističkim svojstvima.

Usklađivanje značajki nije samo zastupljena u okviru problematike sinteze dvodimenzionalnih, već i trodimenzionalnih tekstura. Stoga ćemo većinu naše pažnje usmjeriti upravo njoj.

3.2 Sinteza volumnih tekstura

Sinteza volumnih tekstura predstavlja veliki izazov modernim računalima. Prije nego se upustimo u ovo zanimljivo područje računalne grafike, potrebno je postaviti neke granice što se tiče izbora uzoraka iz kojih se može generirati solidna tekstura. Naime, nije moguće iz svih uzoraka dobiti realističnu solidnu teksturu [2]. Da bismo što jednostavnije dočarali ovaj stav, pogledajmo sliku 3.2.1 a) i b) .



a) b)

Slika 3.2.1 a) uzorak koji se ne može koristiti za generiranje tekstura
b) uzorak pomoću kojeg se mogu sintetizirati kvalitetne teksture

Ako se prisjetimo tvrdnje da sintezom iz nekog uzorka želimo dobiti teksturu koja sliči danom uzorku, tada postaje jasno zašto nije moguće generirati teksture iz svih uzoraka. Intuitivno se također nameće i slijedeća tvrdnja: što je uzorak homogeniji, tj. što su različiti dijelovi uzorka međusobno sličniji, to je sintetizirana tekstura realnija i kvalitetnija.

Iako postoji više različitih implementacija algoritma sinteze hipertekstura, oni se svi temelje na sličnim principima. Ono u čemu se razlikuju jesi uglavnom optimizacijske prirode, dakle nisu direktno vezani za sam algoritam generiranja, već na njega utječu neposredno.

3.2.1 Susjedstva

Susjedstvo jest bilo koji dio uzorka, čiji su oblik i veličina unaprijed definirani. To su najčešće kvadratni ili kružni prozori, čija veličina varira u zavisnosti o uzorku s kojim se radi. Da bismo u potpunosti definirali susjedstvo, moramo napomenuti da se ono uvijek razmatra u smislu nekog piksela. Primjeri različitih susjedstva dani su na slici 3.

Slika 3.2.1.1 a) kauzalno susjedstvo, b) nekauzalno susjedstvo

Na slici *3.2.1.1* također vidimo da se susjedstvo nekog piksela može definirati na različite načine. Primjerice na slici *3.2.1.1* a) vidimo da se promatrani piksel p_i nalazi u središtu svog susjedstva, dok se na slici *3.2.1.1* b) nalazi na kraju. Smisao ovako različitog interpretiranja pojma „susjedstava“ nije bez razloga.

3.2.2. Udaljenost dvaju susjedstva

Sinteza volumne teksture temelji se na slijedećem postupku: za dani uzorak *E* (*od eng. exemplar*) se najprije generira kocka proizvoljnog volumena i proizvoljnih vrijednosti elemenata volumena (voksela od *eng. volume pixel*). Da bi se iz nje generirala volumna tekstura, moramo iterativno promijeniti vrijednost svakog voksela. Postavlja se pitanje kako odrediti u što se pojedini voksel preslikava.

Kada razmatrano uzorak, možemo uočiti da svaki slikovni element ovisi o svojoj okolini, tj. o svome susjedstvu. Na jednak način potrebno je da svaki volumni element unutar volumne teksture ovisi o svome susjedstvu, točnije o njima tri, za svaku os po jedan. Dakle, naš problem određivanja vrijednosti volumnih elemenata se preslikava u problem određivanja njegovih susjedstva te zavisnosti o njima. Međutim, naša volumna tekstura u početku sadrži samo slučajno generirane vrijednosti [1].Drugim riječima, vrlo je mala vjerojatnost da nam se unutar nje nalaze identična susjedstva kao u uzorku *E*.

Intuitivno se može doći do idućeg zaključka: ako ne možemo odrediti vrijednost voksela izravno iz njegovih susjedstva, tada je možemo odrediti iz susjedstava uzorka *E* koji su najsličniji stvarnim susjedstvima promatranog voksela. Ovaj zaključak je ključan za algoritam generiranja volumnih tekstura.

Pojam „sličan“ može se interpretirati na više načina, no najčešći je slijedeći: dva susjedstva to sličnija što im je vektorska udaljenost manja. Matematički zapisano, za RGB model boja bez komponente prozirnosti, te proizvoljne veličine i forme susjedstva, udaljenost tih susjedstva definirana je na sljedeći način:

$$D = \sqrt{\sum_{R,G,B} (s_i - e_i)^2}$$

Općenitije rečeno, svako se susjedstvo zapisuje kao *n* x *k* dimenzionalni vektor, pri čemu *n* označava broj elemenata (slikovnih ili volumnih) koje susjedstvo sadrži, dok *k* predstavlja broj komponenata modela boje kojim je „obojano“ susjedstvo. Udaljenost se dvaju susjedstva definira kao L_2 norma ovakva dva vektora.

3.2.3 Određivanje vrijednosti voksela

Sada kada nam je poznato da svaki voksel ovisi o svojim susjedstvima, potrebno je definirati kako iz tih susjedstava odrediti vrijednost voksela. Na ovom mjestu potrebno je napomenuti da postupak izravno ovisi o tome kakva ćemo susjedstva koristiti. Naime, algoritam neće postići jednake rezultate koristeći kauzalna ili nekauzalna susjedstva.

Kao što je rečeno, ako je susjedstvo voksela kauzalno, tada se on nalazi na posljednjem mjestu susjedstva, a ako je nekauzalno, tada se on nalazi u njezinome središtu. Zašto se voksel nalazi baš na tim položajima unutar susjedstva? Odgovor na ovo pitanje leži u odabiru redoslijeda kojim ćemo obilaziti voksele i mijenjati njihove vrijednosti. Naime, nije svejedno da li ćemo voksele obilaziti redom jedan za drugim, ravninu po ravninu, ili ćemo im pristupi nekim slučajnim redom.

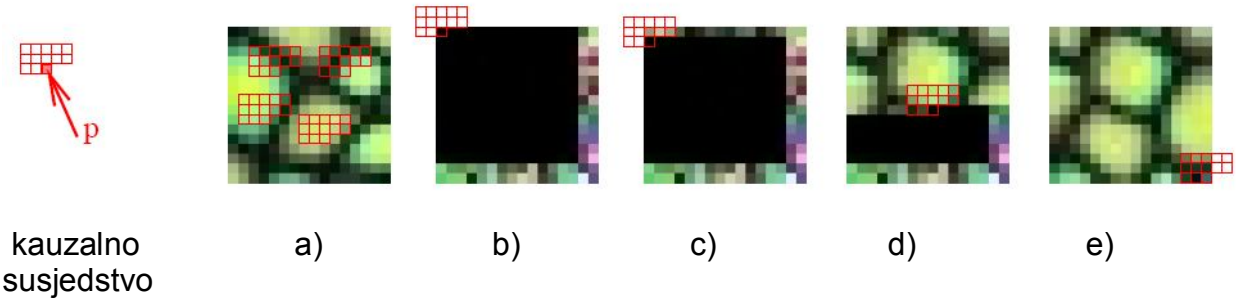
3.2.3.1 Kauzalna susjedstva

Ako voksele obilazimo određenim redoslijedom, primjerice od koordinate (0, 0, 0) do (*n, n, n*), gdje *n* označava duljinu stranice kocke, koristeći kauzalno susjedstvo, naići ćemo na jednu vrlo zanimljivu pojavu. Pretpostavimo da se nalazi na početku našeg algoritma u točki (0, 0, 0). Kako je naša kocka ograničenih dimenzija može pretpostaviti da je njezino se susjedstvo nalazi na suprotnim rubovima generirane kocke. Takvo nešto možemo postići operatorom *modulo n*. Na slici 3.2.3.1 dano je pojašnjenje ove tvrdnje.

Slika 3.2.3.1 a) oblik kauzalnog susjedstva,
b) oblik nekauzalnog susjedstva rubnog elementa

Promotrimo sada sliku 3.2.3.2 . Ona prikazuje kako algoritam određivanja vrijednosti piksela napreduje tijekom vremena. Radi jednostavnosti prikazana je primjena algoritma na generiranje 2D teksture, no princip je isti kao za i generiranje volumne teksture.

Na slici 3.2.3.2 a) prikazan je uzorak *E*. Slika b) prikazuje početno stanje algoritma, pri čemu su pikseli koji će se pokazati nevažnim za naš algoritam obojani crno. Naime, da bismo odredili vrijednost prvoga piksela koristi ćemo susjedstvo koje je označeno na slici 3.2.3.1 b). Slika c) pokazuje stanje algoritma nakon što se generirao prvi redak, a slika d) stanje algoritma kada se već generiralo više od pola teksture. Sada možemo pojasniti zašto su pikseli koji su označeni crno nevažni za naš algoritam. Naime, kao što se vidi iz priloženog, ti pikseli nikada neće biti uzeti u obzir tijekom algoritma jer će se njihova vrijednost sigurno promijeniti prije nego što oni postanu susjedstvo nekog piksela čiju vrijednost računamo [2].

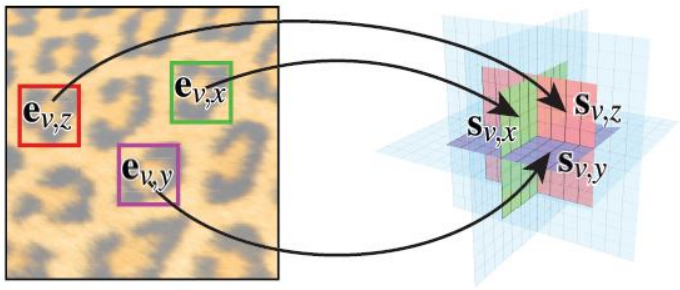


Slika 3.2.3.2 Tijek algoritma sinteze a) uzorak, b) stanje na početku algoritma, c) stanje nakon što je algoritam obradio pri redak, d) stanje kada je obrađena polovina piksela, e) završetak algoritma

Također se sada može shvatiti zašto se ovakvo susjedstvo naziva kauzalno. Naime, prilikom generiranja nove vrijednosti piksela, svi pikseli njegovog susjedstva već su generirani u prethodnim koracima, te se stvorila uzročno-posljedična veza između piksela koji se generira i njegovog susjedstva. Otuda i naziv kauzalno susjedstvo.

3.2.3.2 Nekauzalna susjedstva

Nasuprot kauzalnih susjedstva nalaze se nekauzalna. Njih primjenjujemo ako naš algoritam obilazi voksele nekim slučajnim redom. Kako ne možemo sa sigurnošću znati da su svi vokseli već promijenili svoju vrijednost, tj. da postoji uzročno-posljedična između njih i promatranog voksela *v*, moramo koristiti takvo susjedstvo gdje se voksel *v* nalazi u njegovom središtu. Na slici 3.2.3.2.1 prikazanu su nekauzalna susjedstva voksela te njemu pripadajuća najbliža susjedstva u uzorku *E* [1].



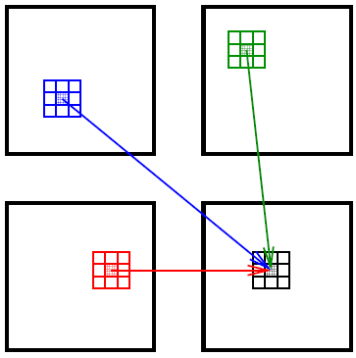
Slika 3.2.3.2.1 Nekauzalna susjedstva volumnog elementa i pripadajuća najbliža susjedstva u uzorku

S (*od eng. solid texture*) označava kocku proizvoljne dimenzije unutar koje želimo generirati našu volumnu teksturu, a $s_{v,x}$, $s_{v,y}$ i $s_{v,z}$ susjedstva trenutno promatranog voksela. Sa *E* označen je uzorak, dok $e_{v,x}$, $e_{v,y}$ i $e_{v,z}$ predstavljaju najbliža susjedstva susjedstvima $s_{v,x}$, $s_{v,y}$ i $s_{v,z}$.

3.2.3.3 Izračun vrijednosti voksela

Nakon što odredimo susjedstva voksela, i njima najbliža susjedstva unutar uzorka, imamo sve što nam je potrebno da bismo izračunali novu vrijednost voksela. Međutim, postoje razni načini kako to učiniti, i svi oni generiraju različite rezultate. Primjerice, u kauzalnim susjedstvima možemo odrediti vrijednost novog voksela *v* kao aritmetičku sredinu RGB vrijednosti voksela v_i, v_j i v_k, pri čemu su to vokseli čiji položaji unutar svog susjedstva odgovaraju položaju voksela *v* (slika 3.2.3.3.1) [2]. Naravno, ne mora se nužno koristiti aritmetička sredina, može to biti i bilo koja druga, primjerice suma RGB vrijednosti tih volumnih elemenata pomnoženih s nekim težinskim koeficijentima w_i , w_j *i* w_k. Ako doduše radimo s nekauzalnim susjedstvima, tada je učinkovitije koristiti takav izračun gdje se cijelo susjedstvo uzima u obzir, a ne samo vokseli v_i , v_j *i* v_k [1].

Naposljetku, da bismo odredili koji će izračun generirati najbolje rezultate, moramo svaki od njih zasebno testirati. Tek onda će nam se pokazati koji od njih daje koliko je koji pristup kvalitetan.



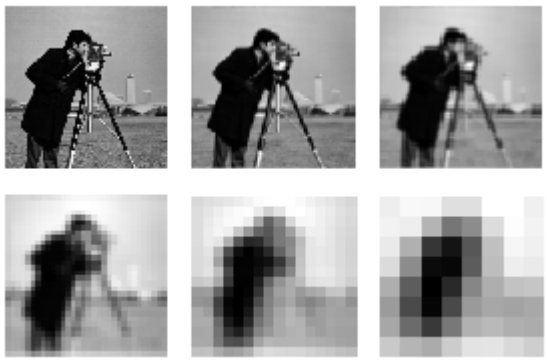
Slika 3.2.3.3.1 Najbliži susjedi. Crvenom, zelenom i plavom označena su najbliža susjedstva u uzorku, dok ispunjeni elementi predstavljaju voksele v_i , v_j i v_k. (Uzorak je naveden 3 puta radi lakše percepcije susjedstva)

4. Gaussove piramide i višerezolucijski algoritam

Uzorci iz kojih se generiraju volumne teksture mogu biti raznih dimenzije, te ne moraju nužno biti homogene. Da bismo iz uzorka generirali kvalitetnu volumnu reprezentaciju, nužno je da veličina susjedstva bude dovoljna da obuhvati i one najveće elemente uzorka (inače se može dogoditi da volumna tekstura bude generirana od samo određenih lokacija uzorka). To za posljedicu, međutim, ima drastično smanjenje performansi algoritma. Kako bi se taj problem izbjegao koriste se više rezolucijski algoritmi, tj. algoritmi kod kojih veličina susjedstva nije konstantna, a temelje se na Gaussovim piramidama.

4.1 Gaussove piramide

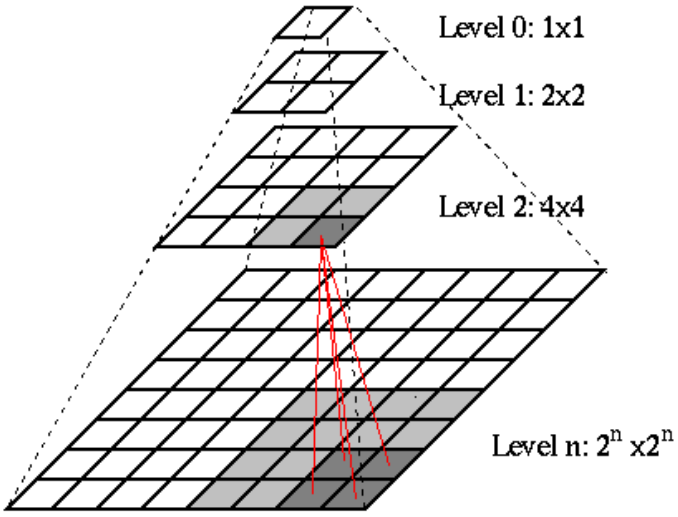
Gaussove piramide predstavljaju tehniku u obradi slika gdje iz početnog uzorka *E*, propuštanjem kroz niskopropusni filtar, dobivaju slike manjih dimenzija. Slike koje se generiraju na ovaj način su manjih dimenzija, mutnijeg sadržaja, i s manjom razinom detalja. Međutim glavna obilježja uzorka ostaju sačuvana. Ova tehnika je poznata i pod imenom Gaussovo zamućivanje (*eng. Gaussian blur*).



Slika 4.1.1 Primjer Gaussovog zamućivanja (postupak proveden 5 puta)

Temelj Gaussove piramide, odnosno n-tu razinu, čini početni uzorak dimenzije $2^n \times 2^n$. Iduća razina generira se iz prethodne propuštanjem kroz niskopropusni filtar koji se naziva Gaussova jezgra. Nakon propuštanja uzorak se skalira na dimenziju $2^{n-1} \times 2^{n-1}$, pri čemu se sadržaj novih slikovnih elemenata

određuje na temelju skupa piksela prethodne razine. Važno je napomenuti da u generiranim razine najvažnija obilježja prethodne razine ostaju sačuvana. To očuvanje je toliko kvalitetno da razine koje su i nekoliko puta manje od početne još uvijek pokazuje veliku sličnost sa njom. Ilustracija ovog postupka prikazana je na slici 4.1.1.



Slika 4.1.1. Gaussova piramida. Nijansama sive označena su analogna područja na različitim razinama.

Na danoj slici može se uočiti još jedna vrlo zanimljiva pojava: propuštanjem slike dimenzije $2^n \times 2^n$ kroz n Gaussovih jezgri dobivamo sliku dimenzije 1×1. Drugim riječima, ova tehnika nam čak omogućava da početni uzorak komprimiramo na jedan jedini slikovni element, tj. piksel. Iako ova krajnost nema praktičnu primjenu, ipak je vrlo važna jer pokazuje kolika je snaga ovog matematičkog aparata.

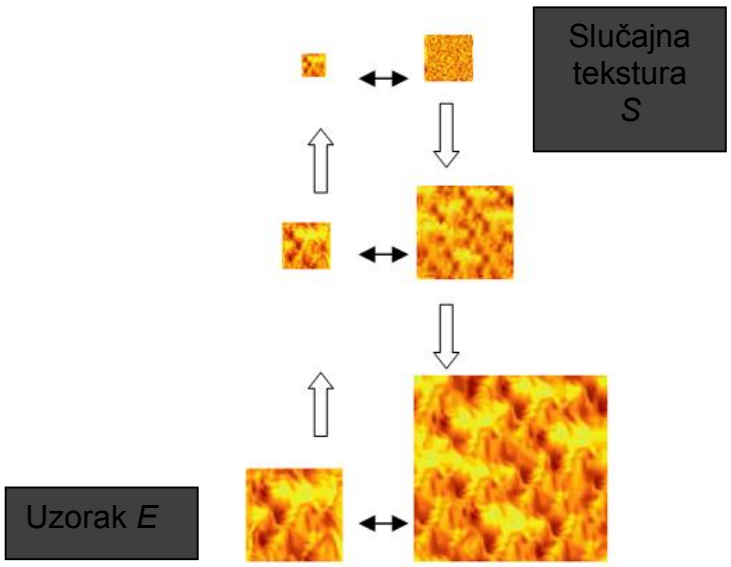
Gaussove piramide, odnosno Gaussova jezgra koja se koristi u procesu njezinog generiranja predstavlja moćno oruđe u područje obrade i kompresije slike. Međutim, ova je tehnika pronašla svoju primjenu i u sintezi tekstura, kako volumnih tako i plošnih.

4.2 Višerezolucijski algoritmi

Višerezolucijski algoritmi [2] predstavljaju skup algoritama za rad sa slikama i plošnim teksturama, i to uglavnom za generiranje njihovih volumnih inačica ili ogromnih plošnih tekstura. Višerezolucijski se u ovom kontekstu odnosi na pretpostavku da veličina susjedstva s kojim algoritmi rade ne mora biti konstanta, već može varirati. Ovi algoritmi nastali su kao rezultat potrebe za optimizacijom prijašnjih algoritama i tehnika koje su se koristili za obavljanje sličnih zadataka.
Prijašnji algoritmi koji su se koristili za generiranje plošnih i volumnih tekstura, primjerice oni opisani u prethodnim poglavljima, jesu davali zadovoljavajuće rezultate. Međutim, kako je složenost i veličina uzoraka rasla, vrijeme potrebno za njihovo izvođenje je postalo neprihvatljivo. Naime, oni su se bazirali na fiksnim dimenzijama susjedstva, tako da su se, u zavisnosti o veličini ulaznog uzorka i njegovoj složenosti, mogli izvoditi i po nekoliko dana koristeći susjedstva dimenzija *32 x 32* i više.

Osnovna prednost višerezolucijskih algoritama jest njihova brzina i, sukladno tome, smanjenja količina računalnih resursa potrebna za njihovo izvođenje (u odnosu na prijašnje algoritme). Ono na čemu se temelji ova optimiziranost jesu Gaussove piramide. Naime, ovi algoritmi ne sintetiziraju volumnu ili plošnu teksturu izravno, tj. prilagođavajući početnu slučajno generiranu teksturu danom uzroku, već prilagođavanje obavljaju iterativno, od najviše do najniže razine.

Postupak je slijedeći: za dani uzorak E se najprije generira višerazinska Gaussova piramida P_E. Nakon toga generira plošna ili volumna tekstura S proizvoljnih dimenzija. Potom se modificiranim postupkom pronalaženja najbližeg susjedstva (objašnjen u nastavku) generiraju nove vrijednosti elemenata teksture S. Nakon što se generira trenutna razina L, provede se obrnuti postupak Gaussovog zamuivanja, te se tako generira nova razina $L + 1$. Postupak se ponavlja onoliko puta koliko P_E ima razina.



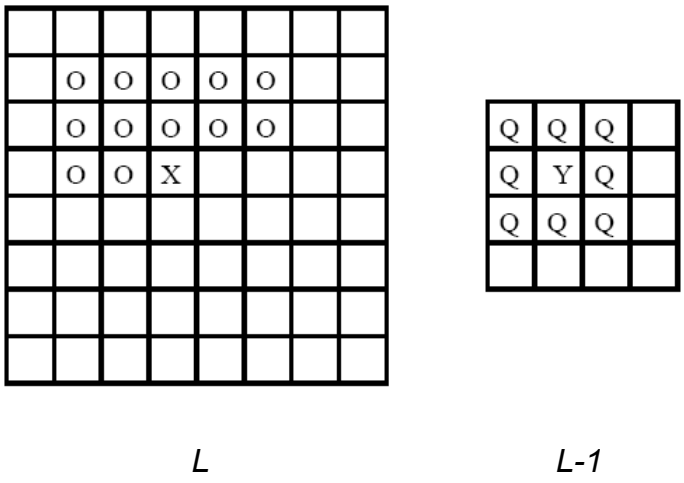
Slika 4.2.1 Primjer generiranja teksture višerezolucijskim algoritmom – bijelim strelicama označen je redoslijed generiranja tekstura, a crnima analogne razine piramida.

4.2.1 Određivanje najbližeg susjedstva u višerezolucijskim algoritmima

U prethodnom poglavlju opisano je okvirno kako se iz danog uzorka generira tekstura. Sada ćemo pobliže opisati kako se u okviru ovog algoritma određuju najbliža susjedstva.

Izraz susjedstvo se u okviru višerezolucijskih algoritama proširuje zbog postojanja dvije Gaussove piramide, P_E i P_S. U poglavlju 3.2.1 definirali smo susjedstvo kao okruženje jednog volumnog ili slikovnog elementa. Sada se to susjedstvo proširuje tako što u obzir uzimamo i analogno susjedstvo koje se nalazi na razini iznad trenutno promatrane u Gaussovoj piramidi [2]. Jedina je iznimka početna, tj. nulta razina koja nema proširenja. Kod nje se promjena vrijednosti pojedinih elemenata odvija jednako kao u početnoj verziji algoritma.

Na slici 4.2.1.1. prikazana je trenutno promatrana razina L kojoj pripada element *X* čiju vrijednost želimo prilagoditi njegovom trenutnom kauzalnom susjedstvu koje čini skup elemenata *O*. No, također želimo i da nova vrijednost elementa *X*, bez obzira da li se radilo o pikselu ili vokselu, očuva svojstva viših razina, jer se u njima nalaze izražajnija obilježja našeg uzorka.



Slika 4.2.1.1. Višerezolucijska susjedstva. Na lijevoj slici prikazano je trenutno susjedstvo elementa X na razini, a na desnoj njegovo proširenje na razini L-1

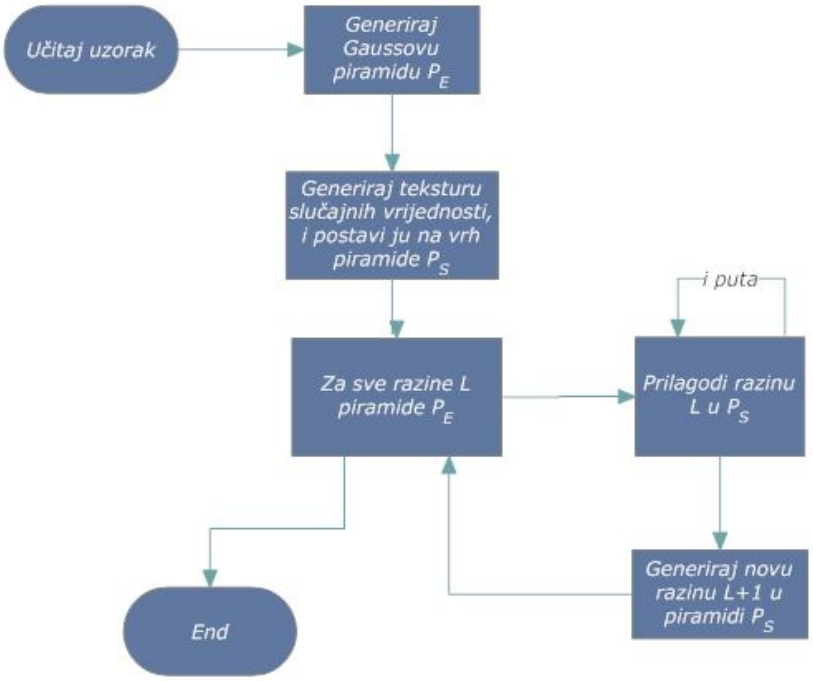
Kako se Gaussova piramida P_S generira od vrha prema dnu, možemo biti sigurni da su više razine već prošle proces optimizacije vrijednosti. Stoga su i sva nekauzalna susjedstva koja se nalaze na razinama višim od trenutno promatrane ispravna, tj. ne sadrže „krive“ elemente. Upravo zbog te činjenice možemo susjedstvo *O* elementa *X* proširiti sa nekauzalnim susjedstvo više razine. To susjedstvo se određuje na temelju koordinata elementa *X*.

Ukratko rečeno, ako su koordinate našeg elementa *X* (*m, n, L*), gdje su m i n koordinate ravnine, a L oznaka razine piramide, tada se kauzalno susjedstvo *O* proširuje s nekauzalnim susjedstvom elementa *Y* koji ima koordinate (*m /2, n /2, L-1*). Ta se struktura u računalu zapisuje kao niz elemenata *O, X, Y* i *Q*.

Sada kada smo odredili susjedstvo (ili susjedstva ako se radi o volumnim teksturama) elementa *X*, preostaje nam da odredimo i njemu najbliže susjedstvo u uzroku. Ono se određuje na potpuno analogan način. Ako trenutno optimiziramo vrijednosti razine L piramide P_S, tada ćemo najbliža susjedstva tražiti također na razini L piramide uzorka P_E (uzevši u obzir i nekauzalna susjedstva na razini L - 1). Nadalje, najbliže susjedstvo *Y* jest ponovno ono koje je po L_2 normi najbliže susjedstvu elementa *X*.

Treba međutim napomenuti da se samo prilagođavanje pojedinih razina ne odvija jedan, već više puta [1], zavisno o razini na kojoj algoritam radi. Primjerice,

za početne razine potreban je veći broj iteracija, jer te razine treba sadržavati najvažnija svojstva uzorka *E*, te je stoga želimo što više prilagodit njemu. Kako se spuštamo po razinama piramide, to naša tekstura *S* sve više i više sliči uzorku *E*, te nam više nije potreban velik broj iteracija da bismo sintetizirali detalje u *S*.



Slika 4.2.1.2 Dijagram toka višerezolucijskog algoritma

5. Programska implementacija

Svi do sada opisani postupci i algoritmi implementirani su programskom jeziku C++. No, prije nego što se posvetimo konkretnoj implementaciji algoritama, usmjerit ćemo našu pozornost na dvije biblioteke koja su uvelike olakšale implementaciju opisanih postupaka.

5.1 Biblioteka EasyBMP

EasyBMP predstavlja biblioteku čiji je kod otvoren (open source) vrste, a namijenjen je za učitavanje, ispisivanje i obradu nekomprimiranih 1, 4, 8, 16, 24 i 32bpp (*eng. bits per pixel*) Windows BMP datoteka. Korišten je u svrhu učitavanja uzoraka u konkretnoj programskoj implementaciji, određivanja susjedstva pojedinih piksel u uzorku te za ispisivanje međurezultata kako bi se ubrzao proces pronalaženja i otklanja pogrešaka.

Zašto je upravo odabran BMP format kao izvor ulaznih podataka? Razlog je u tome što je taj format vrlo fleksibilan, može ga se čitati na gotovo svim sustavima, i jednostavno je organiziran. Upravo zbog ovih svojstava BMP format se pokazao kao idealan izvor podataka, a EasyBMP najboljim dodatakom za rad s njime.

EasyBMP obiluje vrlo korisnim i zanimljivim funkcijama, no u okviru ovoga rada korištene su samo njegove osnovne mogućnosti - čitanja i pisanje BMP datoteka. Čitanje BMP datoteka obavlja se na slijedeći način:

```
BMP inputExemplar;
inputExemplar.ReadFromFile(inputFile);
```

Pisanje u BMP datoteka jest analogno tome:

```
BMP outputExemplar;
outputExemplar.WriteToFile(outputFile);
```

Međutim, dvije najvažnije funkcije ove biblioteke jesu čitanje i zapisivanje vrijednosti piksela u nekoj BMP datoteci.

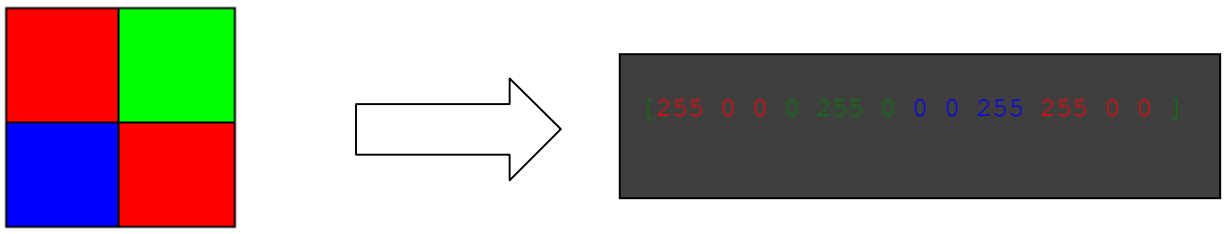
```
inputExemplar.GetPixel(coord_x, coord_y);
outputExemplar.SetPixel(coord_x, coord_y);
```

Funkcija GetPixel() se koristi za dohvaćanje vrijednosti piksela uzroka kako bi se ona mogla pohraniti u jedan prikladniji datotečni format – binarni. Nasuprot tome, funkcija SetPixel() se koristila za iscrtavanje međurezulata. To iscrtavanje je bilo prijeko potrebno kako bi se ustanovilo da li volumna tekstura *S* divergira ili konvergira prema uzroku *E*.

5.2 Biblioteka ANN

ANN jest *približno najbliži susjed* (engl. *ANN je* akronim za approximate nearest neighboor). To je biblioteka koji podržava pretragu za najbližim susjedom u prostorima proizvoljnih dimenzija, pri čemu pretraživanje može biti precizno ili aproksimativno. Naime, za dani skup susjeda (odnosno susjedstva), i jedan ulazni podatak iste dimenzije, ANN vraća 1 ili više najbližih susjedstva. To vraćeno susjedstvo može biti apsolutno najbliže (ako se koristi iscrpna pretraga), ali ne mora. Naime, ANN podržava i modificiranu pretragu koja može vratiti susjedstvo koje nije apsolutno najbliže, ali koje nije udaljeno od stvarno najbližeg za više od nekog faktora *k.*

Kako bi ANN ispravno radio u ovom kontekstu, susjedstva se transformiraju u formu vektora, tako se u vektoru redom navode vrijednosti komponenti boje, ali ne i koordinata. Drugim riječima, susjedstvo $m \times n$ će se, pod pretpostavkom da se koristi RGB model boja, preslikati u vektor duljine $(m*n*3)$. Slika 5.2.1 ilustrira ovu transformaciju.



Slika 5.2.1 Primjer transformacije jednostavnog susjedstva u vektor.

ANN jest vrlo fleksibilan library što se tiče pojma udaljenosti susjedstva. Naime, ANN podržava i promjenu načina izračunavanja udaljenosti, tj. moguće je specificirati koji će se udaljenost koristiti prilikom pretrage za najbližim

susjedstvom. Primjerice, prema standardnim postavkama, udaljenost je definirana kao vektorska, no to može biti i manhattan ili kvadratna udaljenost.

Osnovna struktura podataka koja se koristi u pretrazi najbližeg susjedstva jesu *kd*-stabla. *Kd*-stabla se izgrađuju na temelju poslanog niza susjedstva između kojih se treba odvijati pretraživanje. U kontekstu ovoga rada, taj skup će predstavljati sva susjedstva trenutne razine Gaussove piramide uzorka P_E. Drugim riječima, ako za susjedstvo q (s razine *L* iz P_S) želimo pronaći najbliže susjedstvo na razini L piramide P_E, tada će skup pretraživanja obuhvatiti sva susjedstva dimenzije jednake dimenziji q koja su dio razine *L*.

Definicija konstruktora ove strukture jest slijedeća :

```
ANNkd_tree::ANNkd_tree(
      ANNpointArray pa,       // niz podataka nad kojima se obavlja
                              // pretraga
      int n,                  // broj podataka
      int d);                 // dimenzija
```

Pozivi se nad njome izvršavaju na slijedeći način:

```
kdTree = new ANNkd_tree(      //konstruktor
                 dataPts,     //polje susjedstava
                 nPts,        //broj susjedstava
                 dim);        //dimenzija susjedstva


kdTree->annkSearch(           // pretraži
            queryPt,          // susjedstvo q
            k,                // broj najbližih susjeda koje želimo
                              // odrediti
            nnIdx,            // polje indeksa
            dists,            // polje udaljenosti
            eps);             // dozvoljena pogreška
```

Polje nnIdx će nakon obavljene pretrage sadržavati indekse polja dataPts pri čemu se nnIdx[0] sadrži indeks najbližeg susjedstva, nnIdx[1] indeks drugog najbližeg itd.

Polje dists sadrži udaljenosti susjedstva q od pojedinih susjedstava iz dataPts. U dists[0] se nalazi iznos udaljenost susjedstva q i njemu najbližeg susjedstva iz dataPts, u dists[1] iznos udaljenosti do drugog najbližeg susjedstva itd.

5.3 Aplikacija SolSyn

U ovome dijelu biti će opisane najvažnije funkcije konkretno implementiranoga programskog rješenja. Rješenje se temelji na višerezolucijskom algoritmu, uz napomenu da metode generiranja Gaussovih piramida nisu implementirane, već se koriste već gotove piramide. Algoritam je konstruiran tako da radi s trorazinskom Gaussovom piramidom. Radi jednostavnosti zapisa navede su definicije funkcija, a neki parametri, koji su optimizacijskog tipa, ili se poslani funkciji samo da bi ih ona proslijedila dalje, nisu navedeni.

5.3.1 Faza učitavanja

- GetExemplarData()

```
void GetExemplarData(char* inputFile, char* outputFile, char*
outputFileBinary);
```

Ovo je funkcija koja se prva poziva prilikom pokretanja algoritma. Kao parametre prima ime BMP datoteke uzorka, i imena izlazni datoteka (u .txt i .bin formatu). U izlazne datoteke zapisuje RGB vrijednosti svih piksela.

- GetExemplarDataNeigborhoods()

```
void GetExemplarNeighborhoods(char* imageFileName, char*
binDataFileName, int neighSize);
```

Funkcija koja u danome uzorku određuje susjedstva svakog pojedinog piksela. Kao parametre prima ime datoteke u kojoj su zapisane vrijednosti piksela te generira izlaznu datoteku u kojoj su zapisana sva susjedstva. Ova metoda poziva

se također samo jednom, na početku algoritma kada se nalazimo na vrhu Gaussove piramide.

- GetMultiresolutionExemplarNeighoods()

```
void GetMultiresolutionExemplarNeighoods
(char* fileNameHigher,char* imageFileNameHigher, char*
fileNameLower,char* imageFileNameLower int neighSize);
```

Odmah nakon što se razriješi prva razina piramide, poziva se ova funkcija. Ona dohvaća složena susjedstva uzorka na razinama ispod najviše, i to na način da poziva više jednostavnijih funkcija koje vračaju susjedstva na različitim razinama.

5.3.2 Faze pretraživanja i optimizacije

Tijek programa može se podijeliti u dvije faze, koji se međusobno ciklički nadovezuju. Prvi faza jest faza pretraživanja u kojoj za neki volumni element tražimo njemu najbliže susjedstvo u Gaussovoj piramidi uzorka. Druga faza jest promjena vrijednosti volumnog elementa u zavisnosti o pronađenim najbližim susjedima. Nakon završetka druge faze ponovno započinje prva.

- GetVectorizedNeighborhood2D(), GetVectorizedNeighborhood3D()

```
ANNpoint GetVectorizedNeighborhood2D (int coord_x, int coord_y, char*
fileName, char* imageFileName, int neighSize);

ANNpoint GetVectorizedNeighborhood3D (int coord_x, int coord_y, int
coord_z, axis os, char* fileName, int cubeLength, int neighSize);
```

Ove dvije funkcije pozivaju se u fazi pretraživanja. Pomoću njih određujemo kauzalno susjedstvo nekog elementa. 2D verzija se koristi kada nam je potrebno susjedstvo uzorka, dakle za plošne elemente, a 3D za volumne elemente.

- image_FetchNoncasualNeighoodFromSubLvL(), cube_FetchNoncasualNeighoodFromSubLvL()

```
ANNpoint image_FetchNoncasualNeighoodFromSubLvL(int coord_x, int
coord_y, char* fileName, char* imageFileName, int neighSize):

ANNpoint cube_FetchNoncasualNeighoodFromSubLvL(int coord_x, int coord_y,
int coord_z, axis os, char* fileName, int cubeLength, int neighSize);
```

Nekauzalna susjedstva dobivamo pomoću ove dvije funkcije. Prva dohvaća nekauzalna susjedstvo iz Gaussove piramide uzroka P_E, a iz Gaussove piramide P_S.

Nakon što se odrede kauzalna i nekauzalna susjedstva, oni se spajaju u jednu strukturu, te započinje optimizacijska faza. Optimizacijska faza mijenja vrijednost promatranog volumnog elementa na način da ona bude jednaka srednjoj vrijednosti elemenata najbližih susjedstva koji imaju istu relativnu koordinatu kao i promatrani element.

Ta srednja vrijednost međutim nije aritmetička, jer svako susjedstvo ne utječe na jednak način na promatrani volumni element. Naime, što je neko susjedstvo uzroka bliže susjedstvu promatranog elementa, to ono treba imati veći značaj u konačnoj vrijednosti elementa. Stoga su uvedeni težinski koeficijenti koji su približno jednaki recipročnoj vrijednosti udaljenosti susjedstva.

Međutim, prije nego što se konačno prilagodi vrijednost voksela, potrebno je razmotriti još jedan faktor. Naime, prilikom optimizacije vrijednosti elemenata može se dogoditi da naša generirana tekstura zapadne u jedan lokalni minimum, pa konačna tekstura neće biti slična cijelome ulaznom uzroku, već samo jednom njegovom dijelu. Da bismo to izbjeli konstantno pratimo odnos histograma uzorka i sintetizirane teksture. Ako se pokaže da je došlo do većih razlika u statistikama, tada se prilagođavaju težinski koeficijenti u izračunima vrijednosti elemenata na način da onim elementima koji će pridonijeti slaganju histograma ne mijenjamo težinske koeficijente, a ostalima smanjimo vrijednosti za neki faktor.

Slika 5.3.2.1 Uzorak te generirana tekstura sa i bez praćenja histograma.

Funkcije koje implementiraju netom opisane postupke su slijedeće:

- StartOptimisationFase()

```
void StartOptimizationPhaseV2(int velicina, int cubeLenght, char*
fileName, int neighSize, char* lowerLvlFileName, int lvl);
```

StartOptimisationFase() započinje fazu optimizacije. U njoj se određuje nova vrijednost volumnog elementa prema netom opisanome postupku.

- CalculateDiversionSum()

```
double CalculateDiversionSum (int position, RGB* neighborhood, char*
cubeFile);
```

Ova funkcija vraća broj s kojim treba podijeliti već izračunate vrijednosti težinskih koeficijenata kako bi se prilagodili da generirana tekstura prati statistiku uzroka.

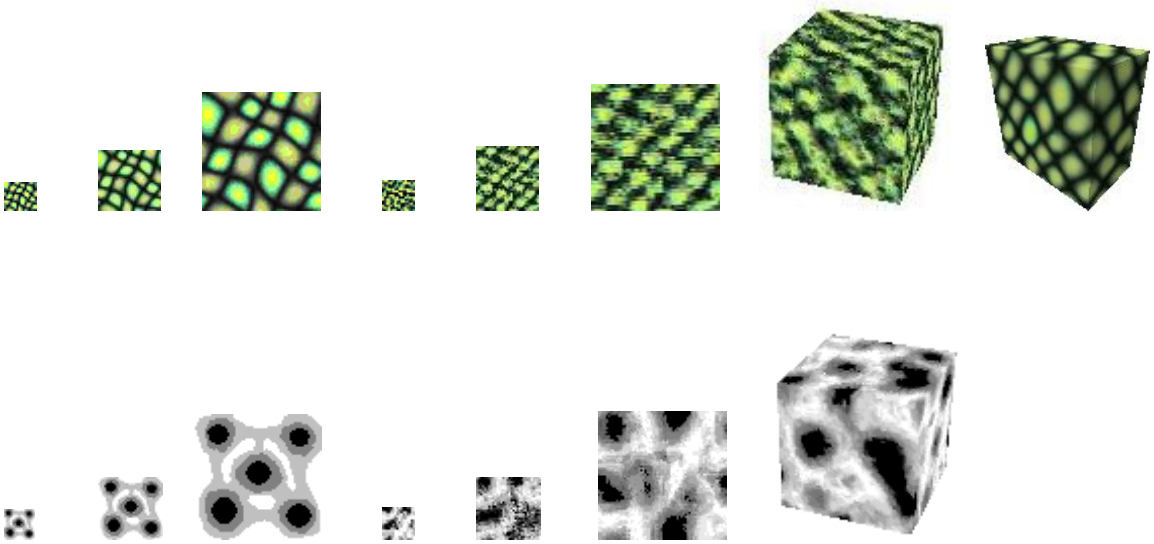
- PovecajRezolucijuInterpoliraj()

```
void PovecajRezolucijuInterpoliraj(char* inputFileName, char*
outputFileName, int cubeLengthInput, int cubeLengthOutput)
```

Ovo je funkcija koja se zadnja izvršava kada se završi jedna razina Gaussove piramide. Ona provodi postupak interpolacije kako bi se generirala jednako vrijedna solidna tekstura, koja je istog sadržaj kao i učitana, samo većih dimenzija.

6. Analiza rezultata

Rezultate koje je dala implementirana programska podrška nažalost nisu bili u skladu sa očekivanjima. U nastavku su priloženi rezultati koje je generirala aplikacija. Najprije su navede sve razine Gaussove piramide uzorka, zatim razine Gaussove piramide dobivene teksture te naposljetku dobivena volumna tekstura. Također je naveden i rezultat za usporedbu, ako je bio dostupan [1].



Slika 6.1 Volumne teksture sintetizirane aplikacijom SolSyn

Iako dobiveni rezultati nisu identični uzorcima, ipak je sličnost između njih očita. Kako bi ovaj algoritam postao što učinkovitiji, potrebno je implementirati još neke dodatne metode. Primjerice, generiranje volumnih tekstura veličine *64x64x64* traje okvirno pet do šest sati rada na 3.2 GHz procesoru, što svakako predstavlja veliki nedostatak. Ovaj problem mogao bi se riješiti implementacijom PCA algoritma (*eng. principal component analysis*) koji dimenzije susjedstva može smanjiti za jedan do dva reda veličine, čime se vrijeme trajanja pronalaženje najbližeg susjedstva drastično smanjuje.

U vidu ovih razmatranja potrebno je također navesti da se Gaussove piramide uzorka nisu generirale tijekom izvođenja programa, već su bile prije

generirane jednim drugim alatom, što je svakako utjecalo na kvalitetu dobivenih uzorka.

7.Zaključak

Generiranje volumnih tekstura i tekstura općenito predstavlja jedan vrlo intrigantan problem iz područja računalne grafike. Teksture čine jednu od glavnih komponenata koje pridonose kvaliteti prikaza, bilo da se radilo o statičkim prikazima ili animacijama.

Nažalost, proces sintetiziranja tekstura jest vrlo zahtjevan i složen. Danas se istražuju algoritmi koji trebaju biti brzi, učinkoviti i primjenjivi na velik raspon različitih uzoraka. Glavne smjernice ovih istraživanja se baziraju na višerezolucijskim algoritmima, superrezolucijama te kompresiji i dekompresiji tekstura.

Stoga je glavni cilj ovoga rada bilo je razmotriti i proučiti postojeće metode sintetiziranja, te pokušati programski implementirati istražene postupke i algoritme. Iako rezultati koje je dala programska implementacija nisu u skladu sa očekivanjima, to ne znači da trebamo odustati od istraživanja ovog područja, već shvatiti to kao poticaj i dodatni izazov.

8. Literatura

[1] Kopf J., Fu C.-W., Cohen-Or D., Deussen O., Lischinski D., Wong T.-T.: *Solid Texture Synthesis from 2D Exemplars,* ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007), 2007., http://www.johanneskopf.de/publications/solid/index.php

[2] Wei L.-Y.: *Texture Synthesis by fixed neighborhood searching*, *studeni 2001*., http://graphics.stanford.edu/papers/liyiwei_thesis/thesis_1side.pdf

[3] Gaussian and Laplacian Pyramids, lipanj 2008., http://gdit.iiit.net/~arul/report/node12.html

9. Sažetak

Postupci sinteze tekstura i teksturiranje objekata

Ovaj rad opisuje osnovne koncepte i algoritme današnjih metoda generiranja tekstura. Rad je podijeljen u šest poglavlja. Prva četiri poglavlja uvode nas u polako u problematiku generiranja volumnih tekstura s teoretskog gledišta. U prvome poglavlju opisuju se osnovni pojmovi vezani uz teksture i njihovu sintezu. Drugo poglavlje posvećeno je slikovnim formatima i modelima boja. Treće poglavlje pruža detaljan uvid u problematiku sintetiziranja tekstura, dok četvrto poglavlje daje pregled optimizacijskih postupaka koji pridonose brzini i kvaliteti generiranja tekstura.

U petom poglavlju dan je opis konkretne programske implementacije, te pripadajućih biblioteka koji se koriste. Šesto poglavlje posvećeno je analizi i usporedbi rezultata.

Ključne riječi: sinteza tekstura, volumne teksture

Abstract

Solid texture synthesis methods and object texturing

This paper describes the basic concepts and algorithms of modern texture synthesis methods. It is divided into six chapters. The first four chapters introduce us with texture synthesis problems from a theoretical point of view. In the first chapter are the basic terms and concepts of textures and texture synthesis described. The second chapter is devoted to image data formats and color models. The third chapter is providing a detailed insight into the problematic of texture synthesis, while in the fourth chapter is a given a list of optimization techniques which ensure fastness and quality.

In the fifth chapter the concrete program application is described, while the sixth chapter is giving an overview of the results.

Key words: texture synthesis, solid texture

Kazalo slika