Sveučilište u Zagrebu

Fakultet elektrotehnike i računarstva

Diplomski rad 1837

Postupci proceduralnog modeliranja gradova tenzorskim poljima

Borko Jandras

Zagreb, 13. ožujka 2010.

Sadržaj

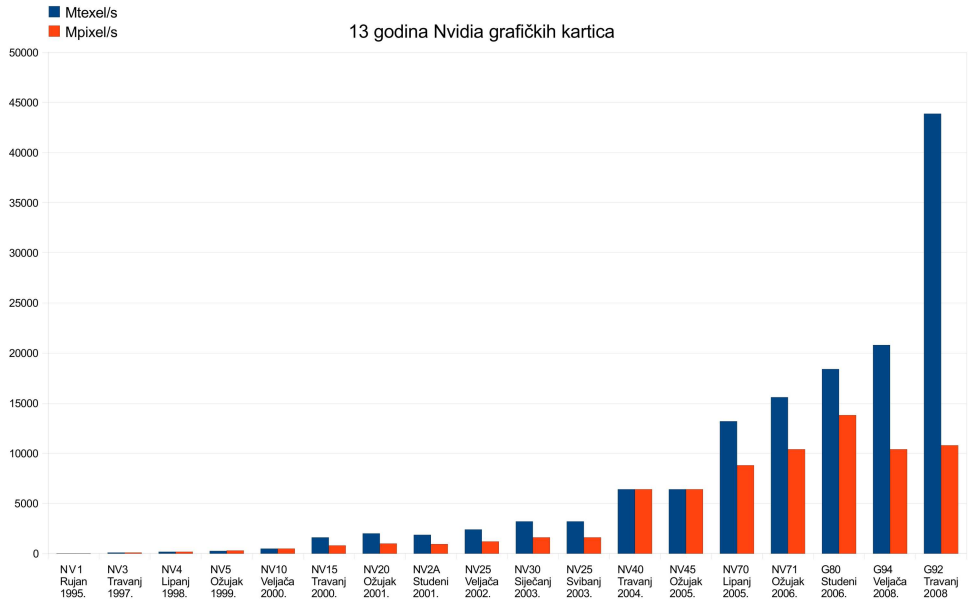
Popis slika

Popis ispisa

1 Uvod

Od samih početaka računalne grafike, sa rastom računalne moći i razvitkom specijaliziranog grafičkog sklopovlja rasla je i mogućnost prikaza sve bogatijih i detaljnijih scena.

Prije pojave grafičkih akceleratora glavni problem bio je kako koristeći raspoloživu računalnu snagu iskoristiti što bolje za prikaz što detaljnijeg prikaza scene. Pojavom grafičkih akceleratora i njihovim ulaskom u široku uporabu, uzrokovano padom cijena istih, bogatstvo i detaljnost scena koje su se mogle iscrtavati u realnom vremenu naglo je porasla (slika 1).



Slika 1: Mogućnosti Nvidia grafičkog sklopovlja kroz godine.
Izvor: Tom's Hardware, 13 Years of Nvidia Graphics Cards

Taj trend u rastu mogućnosti prikaza sve bogatijih scena iskoristili su najviše proizvođači računalnih igara.

Industrija računalnih igara uvijek je bila ta koja je najviše gurala razvoj sve boljeg i sve jačeg grafičkog sklopovlja. Danas se industrija računalnih igara stvarno može nazivati industrijom. Budžeti razvojnih studija narasli su do velikih iznosa. Razvoj moderne računalne igre zahtjeva nekoliko godina i doseže trošak do nekoliko stotina miliuna dolara. Ono što je znakovito je da veći dio tog iznosa odlazi na izradu vizualno sadržaja – modeli, scene, animacije, međusekvence – nego na razvoj programske potpore koja sve to pogoni.

Dok je razvoj grafičkog sklopovlja, algoritama i tehnika donio mogućnost prikaza scena prebogatih detaljima, modeliranje tih scena ostao je isti dugotrajan i mukotrpan postupak kakav je bio i prije deset godina. Računalna moć raste brže nego što raste ljudska sposobnost ručnog modeliranja [1]. Industrija računalnih igara i industrija specijalnih efekata postali su generatori velike potražnje za postupcima što bržeg stvaranja složenih virtualnih okruženja [3].

1.1 Proceduralno generiranje i modeliranje

Ručno modeliranje uistinu je postalo usko grlo u produkciji grafičnih sadržaja. Korištenje proceduralnih tehnika može drastično smanjti vrijeme utrošeno na modeliranje [2].

Proceduralne tehnike su odsječci programskog koda ili algoritmi koji specificiraju neku karakteristiku računalno generiranog modela ili efekta. Na primjer, proceduralno generirana tekstura mramorne površine ne koristi se slikom pravog mramora za određivanje boje pojedinog elementa teksture. Umjesto toga, koriste se algoritmi i matematičke funkcije za određivanje boje pojedinog elementa. Isprva osmišljeni za izradu tekstura za objekte, proceduralno tekstuiranje, sjenčanje i modeliranje postali su sveprisutni, nezaobilazni alati za stvaranje realističnih slika i animacija u aplikacijama od specijalnih efekata do računalnih igara [1].

Proceduralno modeliranje je pojam koji pokriva veći broj tehnika u polju računalne grafike namjenjenih kreiranju modela i tekstura iz skupa pravila. Proceduralno modeliranje se fokusira na stvaranje modela iz zadanog skupa pravila, odmičući od klasičnog postupka stvaranja modela na temelju korisničkog unosa [19].

Da se u proceduralnim tehnikama skrivaju mnoge mogućnosti pokazali su pripadnici "demo-scene", hakerske subkulture specijalizirane za izradu audio-vizualnih prezentacija kojima se pokazuju programerske vještine. U kategoriji "96k", demo programi ograničeni na 96 kB za programski kod i podatake u mogućnosti su prikazati bogate sadržaje koristeći se upravo proceduralnim tehnikama. Računalna igra *.kkrieger*, njemačke demo-grupe *.theprodukkt*, u ukupno 97.280 byte-a spremila je cjelokupnu računalnu igru "first-person shooter" žanra (slika 2). *.kkrieger* se koristi raznim algoritmima proceduralne generacije za spremanje tekstura i geometrije korištene u igri. Riječima autora spomenute igre, *.kkrieger* bi dostizao veličinu od 200 do 300 MB da je sadržaj spremljen na klasičan način.

Osim stvaranja realističnih tekstura prirodnih i umjetnih površina (slika 3), što je do sada bila najčešća primjena proceduralnih tehnika, razvijeni su proceduralni algoritmi za stvaranje raznih atmosferskih efekata poput magle i oblaka, za modeliranje površina vode i terena (slika 4), za modeliranje vegetacije (slika 5), plamena, erozije zemlje, itd.

Često spominjani nedostatak proceduralnog pristupa je određeni gubitak kontrole nad krajnjim rezultatom. Mogućnost stvaranja velike količine detalja ima svoju cijenu – gubitak kontrole korisnika nad tim istim detaljima [1]. No ako krajnji rezultat ne zadovoljava zahtjevima korisnika, onda postupak gubi svoj smisao.

Zahtjev za što većom kontrolom nad krajnjim rezultatom ideja je vodilja razvoja interaktivnih proceduralnih sustava. Napetost između što veće kontrole i programirane kompleksnosti ostaje zanimljiva tema daljnjeg istraživanja [1].

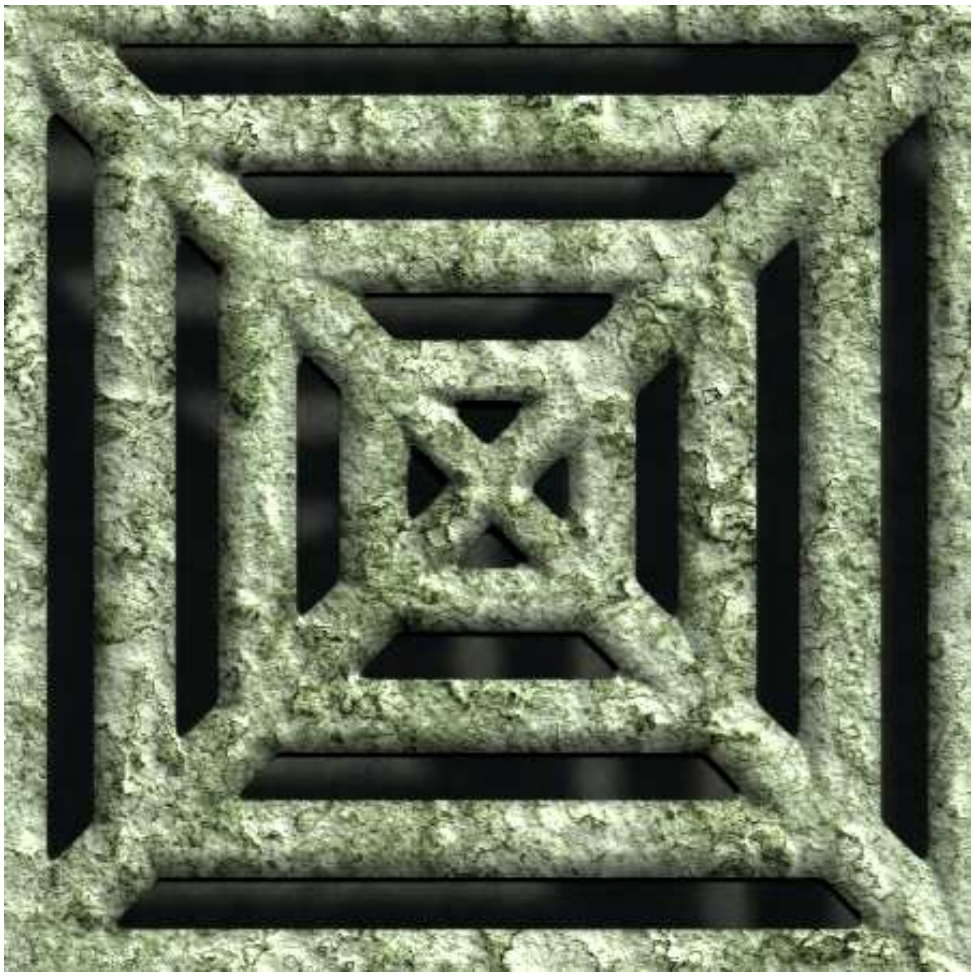
1.2 Proceduralno modeliranje urbanih okruženja

Modeli urbanih sredina – ulica, gradskih četvrti, pa i cjelokupnih gradova pronaze svoju korist u raznim primjenama od računalnih igara, simulatora letenja, industrije specijalnih efekata, pa sve do projekata urbanog planiranja.

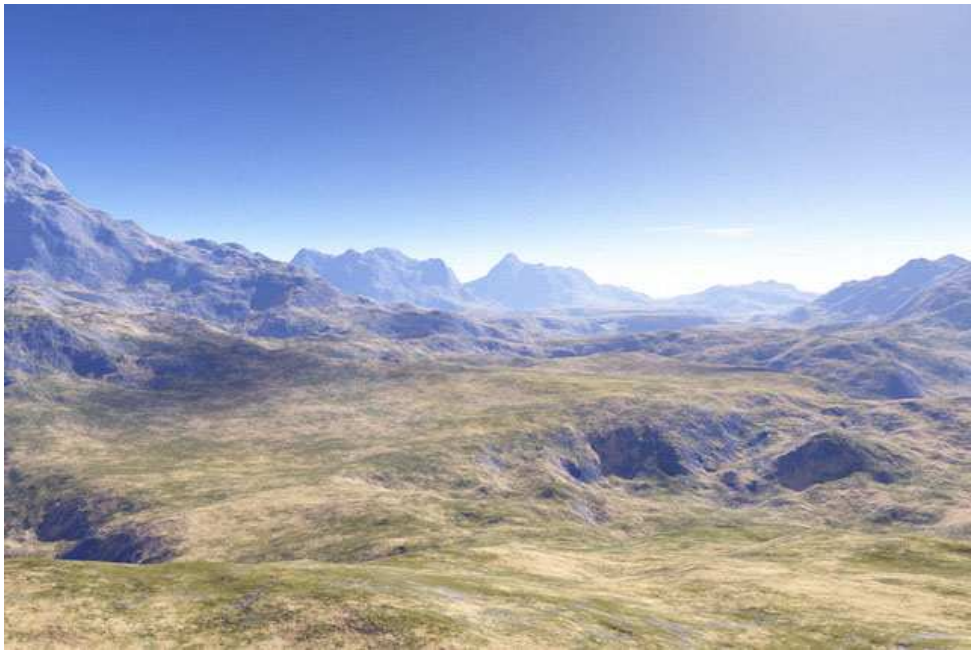
Gradovi su sustavi visoke funkcijske i vizualne kompleksnosti [3]. Model modernog grada može postati iznimno složen ako se pokuša njime vjerno prikazati neki postojeći ili pak detaljno opisati neki fiktivni grad. Mnoštvo razlilitih građevina, zgrada, mostova, ulica i prometnica, etc. rezultiraju mnoštvom detalja koje treba na jedan ili drugi način unijeti u računalni model. Ručno modeliranje svakog djelića virtualnog gradskog okruženja predstavlja izniman poduhvat, pogotovo ako se želi postići živost scene, tj. izbjeći generičnost nastalu ponavljanjem nekoliko građevnih elementata poput zgrada ili ulica.

Primjer detaljno izgrađenog modela jednog fiktivnog grada je računalna igra *Grand Theft Auto IV*, poznatija pod akronimom GTA, koju je razvio studio *Rockstar North* za

Slika 2: Screenshot iz igre .kkrieger



Slika 3: Primjer proceduralne teksture



Slika 4: Fraktalni krajolik
Autor: The Ostrich

Slika 5: Fraktalni korov

izdavača *Rockstar Games*. Na izradu modela fiktivnog grada "Liberty City", u kojemu se odvija radna igre, potrošen je znatan dio ukupnog budžeta od 100 milijuna američkih dolara. Rezultat je doista očaravajuć, pa je igra uspješno vratila uložena sredstva. Gdje god postoji proizvodni proces koji iziskuje velike troškove otvara se prostor inovacijama koje bi taj trošak u nekom omjeru smanjile.

Gradske scene su scene koje podliježu nekim pravilnostima. Postoje uzorci po kojima je nastala mreža gradskih prometnica i ulica. Izgled građevina prati povijesne, estetske i zakonske uvijete [3]. Postoji neki uzorak, ili više njih, po kojima su građene zgrade ili obiteljske kuće. Te činjenice uklapaju se u koncept stvaranja sadržaja iz skupa pravila, što je okosnica svih proceduralnih tehnika.

Priroda terena na kojemu je grad nastao, oblik naseljavanja ljudi u grad, samo su neki od čimbenika koji su uvjetovali trenutnu sliku tog grada. Malo je primjera gradova ili četvrti koji su nastali potpuno planski. Današnji gradovi većim su dijelom nastali nečime što se može opisati kao organski rast.

1.3 Pregled područja

Korištenje proceduralnih tehnika za stvaranje detaljnih modela urbanih okruženja istražili su i u svojim radovima opisali mnogi autori. Slijedeći popis nikako nije iscrpan, već navodi samo one radove koji su najviše utjecali na moje istraživanje ovog uzbudljivog područja računalne grafike.

Parish i Müller su prvi opazili da je modeliranje prometne mreže ključni korak u stvaranju velikog urbanog modela. U svojem radu, *Procedural Modeling of Cities*, Parish i Müller predlažu rješenje za generaciju prometne mreže u obliku otvorenog L-sustava. Ta metoda, koliko god dobre rezultate generirala, ima ozbiljnu manu u vidu nedostatka korisničke kontrole nad krajnjim rezultatom.

Kako bi riješili problem korisničke kontrole, Wonka et.al. u radu *Interactive Procedural Street Modeling* predlažu korištenje tenzorskog polja za navođenje algoritma generatora prometne mreže. Taj pristup omogućuje korisniku da interaktivno uređuje tenzorsko polje koristeći se paletom različitih alata, te da u svakome trenutku ima uvid u efekt kojeg svaka operacija ima na krajnji rezultat [2]. Upravo taj rad Wonke i drugih bio je prvi i glavni uzor za pisanje ovog rada. Tehnike i algoritmi opisani u ovome radu pokušaj su rekreacije tehnika koje su prikazane u *Interactive Procedural Street Modeling*.

Wonka, Wimmer, Silion, Ribarsky u *Instant Architecture* demonstriraju postupak za proceduralno modeliranje elemenata urbane arhitekture. Nacrti zgrada izvedeni su korištenjem *gramatike dijeljenja* (engl. *split grammar*), novog tipa formalne gramatike definirane na konceptu oblika [5].

Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, Luc Van Gool u *Procedural Modeling of Buildings* opisuju *CGA shape* koncept, gramatiku oblika za modeliranje računalno generirane arhitekture, koji proizvodi oblike zgrada visokog nivoa vizualne kvalitete i geometrijskog detalja [4].

Pascal Müller, Gang Zeng, Peter Wonka, Luc Van Gool u *Image-based Procedural Modeling of Facades* opisuju algoritme koji automatski izvode 3D modele visoke vizualne kvalitete iz fasadnih slika proizvoljne rezolucije [6].

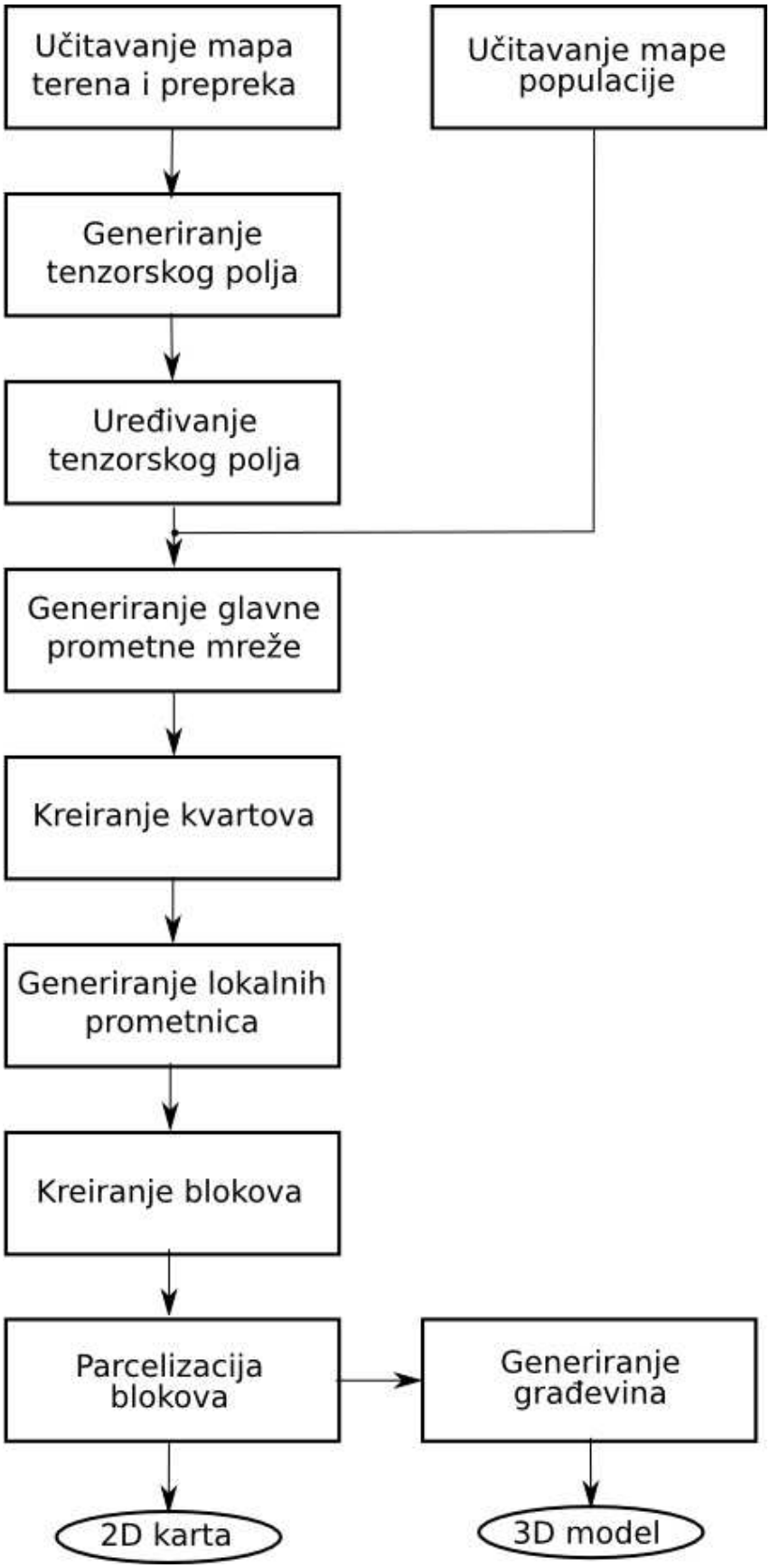
1.4 Pregled postupaka

U ovome radu biti će prikazano kako su uzorci prisutni u izgradnji nekog modernog grada iskorišteni kao pravila algoritama i proceduralnih tehnika koje će pokušati izgraditi računalni model jednog takvog grada. Opisat će se niz tehnika korištenih za modeliranje mnogih aspekata urbanih scena – prometnica, zgrada, kuća, itd.

Svako urbano područje sadrži prometnu infrastrukturu koja prati utjecaje populacije i okoliša [3]. Ako krenemo od ideje da grad raste uzduž i okolo mreže prometnica i ulica koje se protežu nekim terenom, nećemo puno pogriješiti. Oblik mreže prometnica uvijetovat će nekoliko faktora: oblik terena, položaj voda i drugih prirodnih prepreka, gustoća naseljenosti.

Za izradu virtualnog grada potrebno je izraditi prometnu mrežu i generirati veći broj modela građevina. Iz nekoliko slikovnih mapa danih na ulazu u sustav, sustav generira mrežu prometnica i ulica, dijeli zemlju u parcele i kreira prikladnu geometriju za zgrade postavljene na te parcele.

Cijelokupni proces stvaranja virtualnog grada na temelju ulaznih podataka može se prikazati cijevovodom u kojemu je izlaz iz prethodne faze ulaz u slijedeću. Na temelju ulaznih mapa prirodnih prepreka i elevacije terena stvara se tenzorsko polje koje se naknadno može dodatno urediti korisnički zadanim uzorcima. Stvoreno tenzorsko polje ulaz je u fazu generiranja glavne prometne mreže – većih gradskih prometnica. U slijedećog fazi na temelju kreirane glavne prometne mreže obavlja se stvaranje kvartova – zatvorenih područja koje omeđuju prometnice glavne prometne mreže u kombinaciji sa granicama prirodnih prepreka. Unutar pojedinih kvartova identičnim postupkom kojime je stvorena glavna prometna mreža generira se mreža lokalnih kvartovskih ulica. Ponovnim detektiranjem zatvorenih područja, ovaj puta unutar svakog pojedinačnog kvarta, kreiraju se individualni blokovi – najmanje jedinice građevinskog terena omeđene ulicama. Blokovi se u postupku parcelizacije dijele u zasebne građevinske parcele. Lista građevinskih parcela ulaz je u poslijednju fazu – fazu generiranja geometrije, gdje se na svakoj parceli kreira po jedna građevina. Na slici 6 shematski je prikazan cijelokupni proces.



Slika 6: Shematski prikaz procesa

2 Tenzorska polja

U području računalne grafike polja simetričnog tenzora drugog reda koriste se u mnogim primjenama. Na primjer u nefotorealističnom "slikarskom" renderiranju orijentacije poteza kista navode se tenzorskim poljem koje je okomito na polje gradijenta slike. U tzv. šrafuljnim ilustracijama (linijskim teksturama) glatkih površina, šrafure uglavnom prate jedan od principijelnih smjerova tenzora zakrivljenosti (engl. *curvature tensor*). Kada se radi o reprezentaciji prirodnih smjerova na slici ili na trodimenzionalnom obliku, tenzorska polja omogućuju korištenje puno raznovrsnijeg skupa vizualnih elemenata nego što to omogućuju vektorska polja [11].

U ovome radu tenzorska polja koriste se za navođenje postupka generiranja prometne mreže.

2.1 Matematičke definicije

U fizici i matematici, *tenzor* je poopćenje skalara i vektora, te se, poput vektora, sastoji od više skalarnih vrijednosti [20]. *Rang* tenzora je dimenzija niza potrebnog za njegovu reprezentaciju. Tako je skalar tenzor nultog ranga, vektor je tenzor ranga 1, dok tenzor ranga 2 predstavlja matricu. Linearna transformacija (koja se prikazuje matricom) također je tenzor ranga 2.

Za vektor $\vec{x} \neq 0$ kažemo da je *svojstveni vektor* linearne transformacije L ako za neku skalarnu vrijednost λ vrijedi: $L(\vec{x}) = \lambda\vec{x}$. Vrijednost λ naziva se *svojstvena vrijednost* od L koja odgovara svojstvenom vektoru $\vec{x}$. Iz ove definicije izravno slijedi da je svaki vektor $\vec{y} = \rho\vec{x}, \rho \neq 0$ također svojstveni vektor od L.

Za tenzor $[T_{i,j}]$ kažemo da je *simetričan* ako i samo ako vrijedi $T_{i,j} = T_{j,i}$. Za svaki simetrični tenzor drugog reda postoji jedinstveni rastav na zbroj njegovog izotropnog dijela S i anizotropnog dijela A:

$$T = S + A = \lambda I + \mu \begin{bmatrix} \cos 2\theta & \sin 2\theta \\ \sin 2\theta & -\cos 2\theta \end{bmatrix}, \mu \geq 0, \theta\epsilon[0, 2\pi\rangle.$$

Tenzor A ima svojstvene vrijednosti $\pm\mu$ te dva skupa svojstvenih vektora. *Glavni svojstveni vektori* tenzora A su:

$$\left\{ \rho \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} | \rho \neq 0 \right\},$$

i odgovaju svojstvenoj vrijednosti $+\mu$, dok su *sporedni svojstveni vektori*:

$$\left\{ \rho \begin{bmatrix} \cos(\theta + \frac{\pi}{2}) \\ \sin(\theta + \frac{\pi}{2}) \end{bmatrix} | \rho \neq 0 \right\}.$$

i odgovaraju svojstvenoj vrijednosti $-\mu$. Glavni i sporedni svojstveni vektori simetričnog tenzora drugog reda uvijek su međusobno ortogonalni.

Tenzorsko polje T je neprekinuta funkcija koja svakoj točki $\mathbf{p} = (x, y)\epsilon\mathbb{R}^2$ pridružuje tenzor $T(\mathbf{p})$. Kažemo da je $\mathbf{p}$ *degenerirana* ako $T(\mathbf{p}) = 0$, inače kažemo da je $\mathbf{p}$ *regularna*.

Hiperlinija toka Γ tenzorskog polja T je krivulja koja u svakoj svojoj točki $\mathbf{p} = (x, y)\epsilon\Gamma$ ima tangentu u smjeru svojstvenog vektora tenzora $T(\mathbf{p})$. Tako za anizotropno tenzorsko polje $A(\mathbf{p})$ postoje dvije familije hiperlinija toka – glavna i sporedna, koja odgovara glavnom odnosno sporednom polju svojstvenih vektora.

U nastavku teksta pod pojmom *tenzorsko polje* smatra se **polje anizotropnog dijela simetričnog tenzora drugog reda**.

2.2 Vizualizacija

Kako bi korisnik dobio bolji dojam o utjecaju ulaznih podataka na rezultat faze generacije prometne mreže, te kako bi u svakome trenutku imao općeniti dojam o izgledu polja, potrebno je tenzorsko polje nekako korisniku prikazati.

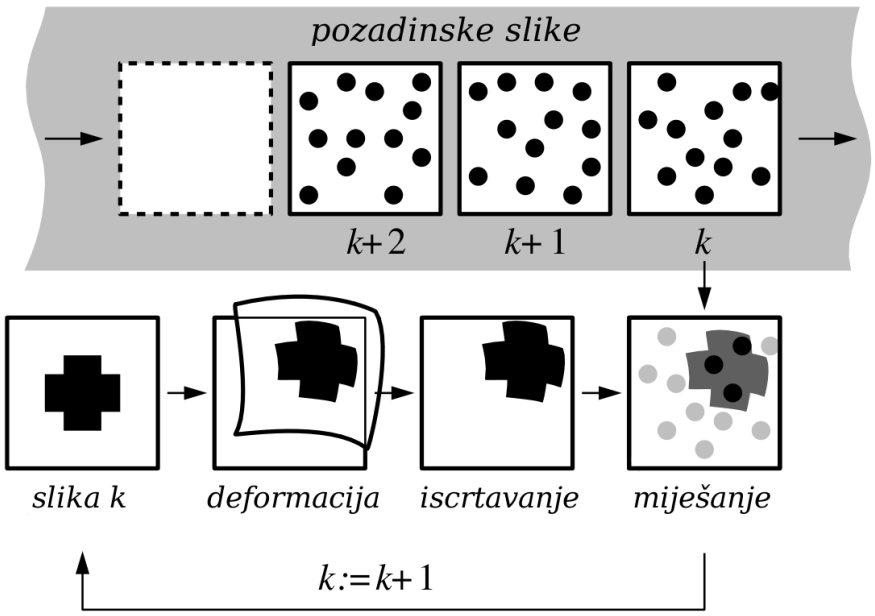
Za vizualizaciju tenzorskog polja razvijene su mnoge metode. Jedan dio metoda bazira se na iscrtavanju pojedinih linija toka. Taj pristup nosi sa sobom veliki nedostatak – krajnji rezultat vizualizacije jako ovisi o pozicijama početnih točaka linija, pa je moguće da neki važni aspekti polja ostanu neprimjećeni [12].

Drugi dio tehnika bazira se na korištenju tekstura za prikaz polja. Najpoznatija tehnika iz te kategorije je metoda *Line Integral Convolution* (LIC) Cabrala i Leedoma. Ta tehnika bazira se na konvoluciji teksture šuma duž linija toka koje izviru iz svakog pojedinačnog piksela. LIC metoda daje dobar rezultat u vidu kvalitete slike, no velik nedostatak joj je znatna računska zahtjevnost.

Za vizualizaciju dvodimenzionalnog toka fluida Jarke J. van Wijk, u svojem radu *Image Based Flow Visualization*, opisuje tehniku simuliranja kretanja i raspada čestica boje u fluidu koja je istovremeno efikasna, računski nezahtjevna, te jednostavna za implementaciju [12].

Vizualizacija polja slikama (IBFV)

Metoda se zasniva na jednostavnom konceptu: svaka slika u slijedu animacije je rezultat miješanja deformirane prethodne slike i neke pozadinske slike, najčešće slike bijelog šuma (slika 7). Proces se može ubrzati korištenjem grafičkog sklopovlja.



Slika 7: Vizualizacija toka korištenjem slika.
Izvor: Image Based Flow Visualization, Jarke J. van Wijk

Neka je zadano vektorsko polje $V(\mathbf{x}) \in \mathbb{R}^2$ koje predstavlja brzinu toka ili smjer i

snagu:

$$V(\mathbf{x}) = \begin{bmatrix} v_x(x,y) \\ v_y(x,y) \end{bmatrix}, \mathbf{x} \in S, S \subset \mathbb{R}^2$$

Linija toka dobije se praćenjem pozicije neke čestice u polju. Za zadanu početnu točku $\mathbf{p}(0)$, jednadžba linije toka je rješenje diferencijalne jednadžbe:

$$\frac{d\mathbf{p}(t)}{dt} = V\Big(\mathbf{p}(t)\Big)$$

Aproksimacijom prvog reda dobije se izraz:

$$\mathbf{p}_{k+1} = \mathbf{p}_k + V(\mathbf{p_k})\Delta t, k \in \mathbb{N}$$

Zamislimo vremenski promjenljivo polje $F(\mathbf{x};t)$ koje predstavlja neko svojstvo koje je provođeno tokom. U našem slučaju polje F predstavlja sliku, pa je $F(\mathbf{x};t)$ tipično RGB trojka. Svojstvo polja F provodi se poput čestice, pa je $F(\mathbf{p}(t);t)$ konstanta duž linije toka $\mathbf{p}(t)$. Koristeći se aproksimacijom prvog reda toka $\mathbf{p}(t)$ polje F može se prikazati izrazom:

$$F(\mathbf{p}_{k+1}; k+1) = \begin{cases} F(\mathbf{p}_k; k) & \mathbf{p}_k \in S \\ 0 & \mathbf{p}_k \notin S \end{cases}$$

Prethodni izraz kaže da se element polja $F(\mathbf{p}_{k+1}; k+1)$ postavlja u crnu boju (vrijednost 0) ako brzina u prethodnoj točki $\mathbf{p}_k$ nije definirana. Prije ili kasnije veći dio slike $F(\mathbf{x};k), \mathbf{x} \in S$ biti će crn, pa će za velik broj točaka $\mathbf{p}_k \in S$ odgovarajuće početne točke $\mathbf{p}_0$ biti izvan područja S. Kako bismo to spriječili u svakom koraku k uzima se kombinacija slike F i neke druge slike G:

$$F(\mathbf{p}_k; k) = (1-\alpha)F(\mathbf{p}_{k-1}; k-1) + \alpha G(\mathbf{p}_k; k),$$

gdje je $\alpha \in [0,1]$ faktor miješanja dviju slika. Za sliku G najčešće se uzima slika bijelog šuma.

Ako promatramo što se događa sa pojedinim elementom slike F – točkom koja se kreće kroz polje, možemo primjetiti slijedeće: točke se kreću kroz tok i s vremenom nestaju. Jobard et al. daju fizičku interpretaciju: čestica se giba dvodimenzionalno i istovremeno tone konstantnom brzinom u poluprozirnom fluidu.

Postupak

Za pozadinske slike G koristi se niz od m slika $G_i, i = 0, ..., m-1$. Za spremanje slike F koristi se spremnik slike (*framebuffer*), dok se pozadinske slike spremaju u memoriju za teksture (*texture memory*). Za animiranje slike F koristi se teselacija područja S pravokutnom mrežom R (slika 8).

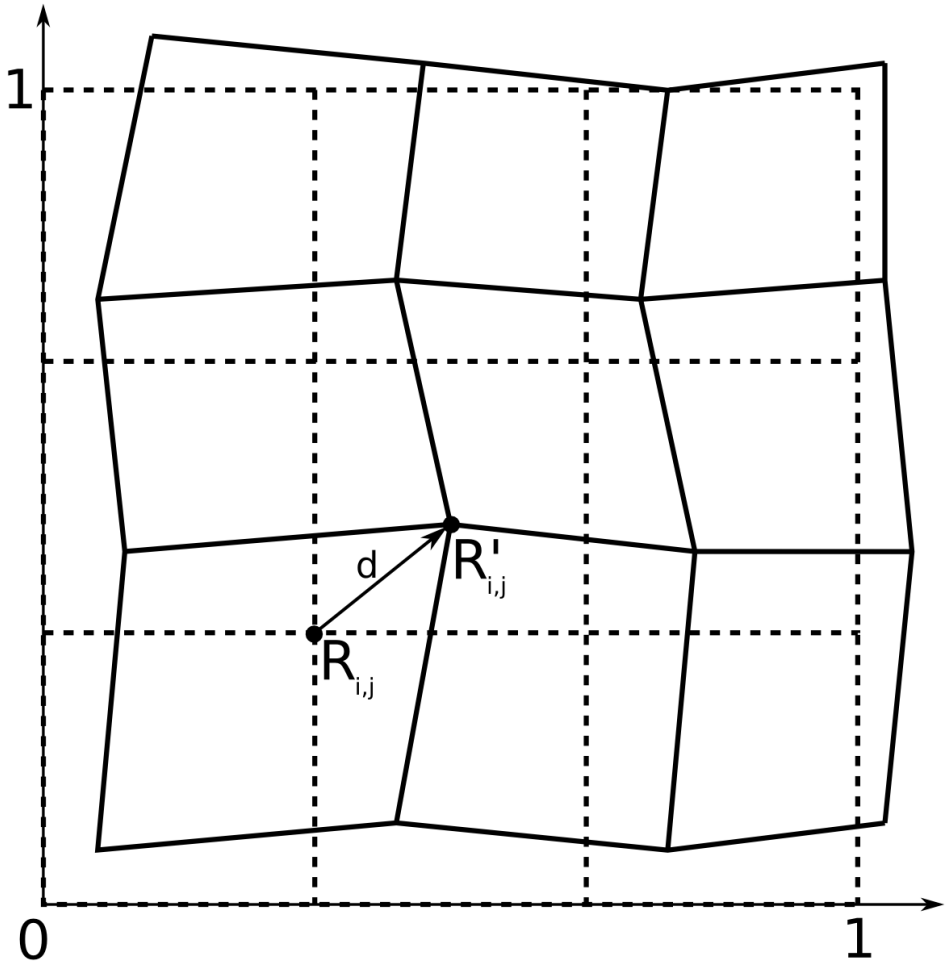
Za generiranje k-te slike slijeda animacije koristi se slijedeći postupak:

1. Ako se polje V promijenilo računaju se koordinate deformirane pravokutne mreže *R'*. Izračun deformirane pravokutne mreže obavlja se po vrhu $R_{i,j}$, gdje se računa pomak vrha (slika 8):
$$d = V(R_{i,j}), R'_{i,j} = R_{i,j} + d$$
2. Pravokutna mreža *R'* se iscrtava na ekran koristeći prethodnu sliku animacije kao teksturu. Koordinate teksture postavljaju se jednoliko rasprostranjene u intervalu $[0,1]$.

3. Slika pozadinskog šuma miješa se sa trenutnom slikom. Preko trenutne slike iscrtava se pravokutnik sa pozadinskom slikom $G_i, i = k\%M$ zadanom kao teksturom, koristeći faktor α kao koeficijent miješanja.

4. Rezultantna slika sprema se u memoriju za teksture. Tamo je spremna za generiranje slijedeće slike u slijedu.

5. Iscrtavaju se ostali elementi slike.



Slika 8: Deformacija pravokutne mreže.
Isprekidane linije: pravokutna mreža R.
Pune linije: deformirana pravokutna mreža R'.

Prilagodba za vizualizaciju tenzorskih polja

U svojem radu *Interactive Tensor Field Design and Visualization on Surfaces* [11], Zhang et al. predlažu ekstenziju IBFV tehnike za vizualizaciju tenzorskih polja.

Za vizualizaciju tenzorskog polja T dovoljno je vizualizirati polje njegovog glavnog karakterističnog vektora. Pri tome nema gubitka informacije jer je polje sporednog karakterističnog vektora identično polju glavnog zakrenutog za 90° [11].

Za vizualizaciju polja glavnog karakterističnog vektora tehnikom IBFV poželjno je polje prethodno pretvoriti u kontinuirano vektorsko polje V. Za to je dovoljno odabrati jednaku orijentaciju za sve točke domene kako bi se otklonila dvoznačnost smjera nastala pri pretvorbi tenzorskog polja u vektorsko polje.

Neka je D domena polja i neka je $S(V) \subset D$ skup točaka u kojima je V nekontinuirano. Konstruiraju se dva vektorska polja, V_1 i V_2, takva da $S(V_1) \cap S(V_2)$ sadrži samo degenerirane točke tenzorskog polja T dok je svaka regularna točka $\mathbf{p} \in D$ sadržana u $D \setminus S(V_i)$. Polje glavnog karakterističnog E vektora može se definirati preko dva prostorno promijenljiva skalarna polja ρ i θ:

$$E = \pm\rho \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix}, \rho \geq 0$$

Definiramo vektorska polja:

$$V_1 = V_x = \begin{cases} \rho \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} & \cos\theta \geq 0 \\ \rho \begin{bmatrix} -\cos\theta \\ -\sin\theta \end{bmatrix} & inace \end{cases}$$

$$V_2 = V_y = \begin{cases} \rho \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} & \sin\theta \geq 0 \\ \rho \begin{bmatrix} -\cos\theta \\ -\sin\theta \end{bmatrix} & inace \end{cases}$$

Dakle, polje V_x se dobije iz polja E birajući smjerove tako da je x-komponenta od V_x svugdje ne-negativna, tj.:

$$S(V_x) = \{(x, y) | \cos\theta_{x,y} = 0\}$$

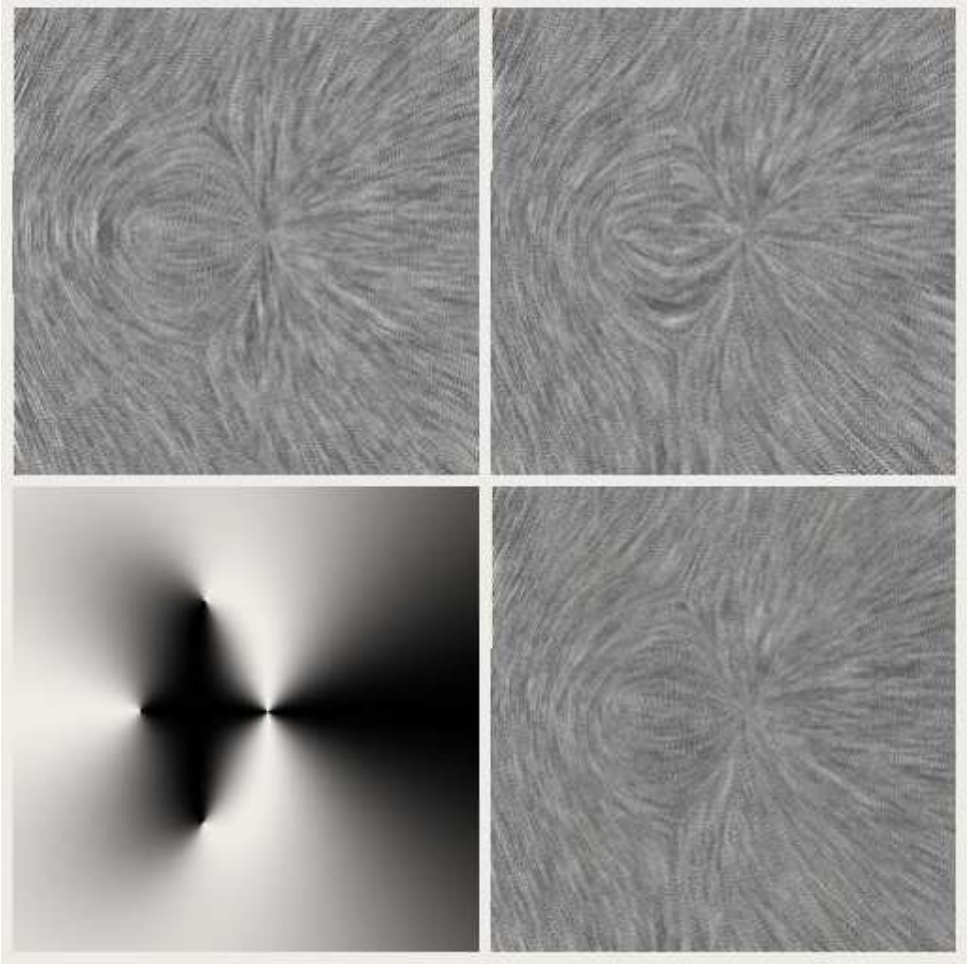
Slično, V_y je dobiven birajući smjerove tako da je y-komponenta od V_y ne-negativna, tj.:

$$S(V_y) = \{(x, y) | \sin\theta_{x,y} = 0\}$$

Definiramo I_x i I_y kao slike dobivene korištenjem tehnike IBFV nad vektorskim poljima V_x i V_y. Neka su $w_x = \cos^2\theta$ i $w_y = \sin^2\theta = 1 - W_x$ funkcije miješanja. Tada završna slika:

$$I = w_x I_x + w_y I_y$$

daje konačnu sliku tenzorskog polja (slika 9).



Slika 9: Vizualizacija tenzorskog polja.
Gore lijevo: slika I_x. Gore desno: slika I_y.
Dolje lijevo: funkcija miješanja w_x.
Dolje desno: završna slika $I = w_x I_x + (1 - w_x) I_y$

Implementacija

Veći, a također i procesorski zahtjevniji, dio postupka može se implementirati koristeći isključivo OpenGL pozivima, te na taj način izvesti korištenjem isključivo grafičkog sklopovlja. Na centralnom procesoru ostaje računje pomaka vrhova pravokutne mreže i generacija pozadinskih slika.

U programskom ispisu 1 naveden je izgled programskog sučelja modula za vizualizaciju tenzorskog polja implementiranog u *C++* jeziku korištenjem OpenGL grafičkog sučelja. Funkcija `paintGL` izvršava se po dva puta za svaku sliku u slijedu animacije – jednom za stvaranje slike I_x, drugi puta za stvaranje slike I_y. Funkcija iscrtava deformiranu pravokutnu mrežu koristeći unaprijed izračunate koordinate vrhova u funkciji `makeMesh`. Preko iscrtane pravokutne mreže miješa se jedna od `NPAT` pozadinskih slika. Pozadinske slike unaprijed se računaju u funkciji `makePatterns` te se prebacuju u memoriju za teksture kako bi bile spremne za korištenje.

```
class FieldPainter
{
public:
    void initializeGL();
    void paintGL(unsigned int frame, bool vx);

private:
    typedef std::vector<Vector2f> Verticen;

    Verticen m_meshTexCoords;    // (u,v) koordinate u vrhovima
    Verticen m_meshVertexVx;     // koordinate vrhova za Ix sliku
    Verticen m_meshVertexVy;     // koordinate vrhova za Iy sliku

    void makeMesh();
    void makePatterns();

    // Konstante.
    //
    static int   const NPN    = 256;
    static int   const NMESH  = 100;
    static float const DM     = 1.0f/(NMESH-1);
    static int   const NPAT   = 32;
    static int   const width  = 257;
    static int   const height = 257;
    static int   const alpha  = 0.12*255;
};
```

Ispis 1: Sučelje modula za vizualizaciju tenzorskog polja

```
void FieldPainter::initializeGL()
{
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
    glEnable(GL_TEXTURE_2D);
    glShadeModel(GL_FLAT);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 1.0, 0.0, 1.0);

    glClear(GL_COLOR_BUFFER_BIT);

    makeMesh();
    makePatterns();
}

void FieldPainter::paintGL(unsigned int frame, bool vx)
{
    Verticen::iterator tex = m_meshTexCoords.begin();
    Verticen::iterator vrt =
        vx ? m_meshVertexVx.begin() : m_meshVertexVy.begin();

    for (int i = 0; i < NMESH-1; i++)
    {
        glBegin(GL_QUAD_STRIP);
        for (int j = 0; j < NMESH; j++)
        {
            glTexCoord2fv(*tex++); glVertex2fv(*vrt++);
            glTexCoord2fv(*tex++); glVertex2fv(*vrt++);
        }
        glEnd();
    }

    glCallList(frame % NPAT); // postavi pozadinsku sliku za teksturu

    // nacrtaj jedinicni pravokutnik
    //
    glEnable(GL_BLEND);
    glBegin(GL_QUAD_STRIP);
    glTexCoord2f(0.0f, 0.0f); glVertex2f(0.0f, 0.0f);
    glTexCoord2f(0.0f, 1.0f); glVertex2f(0.0f, 1.0f);
    glTexCoord2f(1.0f, 0.0f); glVertex2f(1.0f, 0.0f);
    glTexCoord2f(1.0f, 1.0f); glVertex2f(1.0f, 1.0f);
    glEnd();
    glDisable(GL_BLEND);

    // prebaci zavrsnu sliku u memoriju za texture
    //
    glCopyTexImage2D(
        GL_TEXTURE_2D, 0, GL_RGB, 0, 0, width, height, 0);
}
```

U `makeMesh` koristi se funkcija `getV`, koja za zadanu točku $\mathbf{p}$ vraća vrijednosti vektorskih polja $\vec{v}_x = V_x(\mathbf{p})$ i $\vec{v}_y = V_y(\mathbf{p})$. Te vrijednost koriste se za pomak vrhova pravokutne mreže. Vrijednost texturnih koordinata (u, v) ostaju nepromijenjene.

```
void FieldPainter::makeMesh()
{
    m_meshTexCoords.clear();
    m_meshVertexVx.clear();
    m_meshVertexVy.clear();

    Vector2f p1(0, 0);
    Vector2f p2(DM, 0);

    for (int i = 0; i < NMESH-1; i++)
    {
        p1(1) = p2(1) = 0;

        for (int j = 0; j < NMESH; j++)
        {
            Vector2f vx, vy;

            getV(p1, vx, vy);
            m_meshTexCoords.push_back(p1);
            m_meshVertexVx.push_back(p1 + vx);
            m_meshVertexVy.push_back(p1 + vy);

            getV(p2, vx, vy);
            m_meshTexCoords.push_back(p2);
            m_meshVertexVx.push_back(p2 + vx);
            m_meshVertexVy.push_back(p2 + vy);

            p1(1) += DM;
            p2(1) += DM;
        }

        p1(0) += DM;
        p2(0) += DM;
    }
}
```

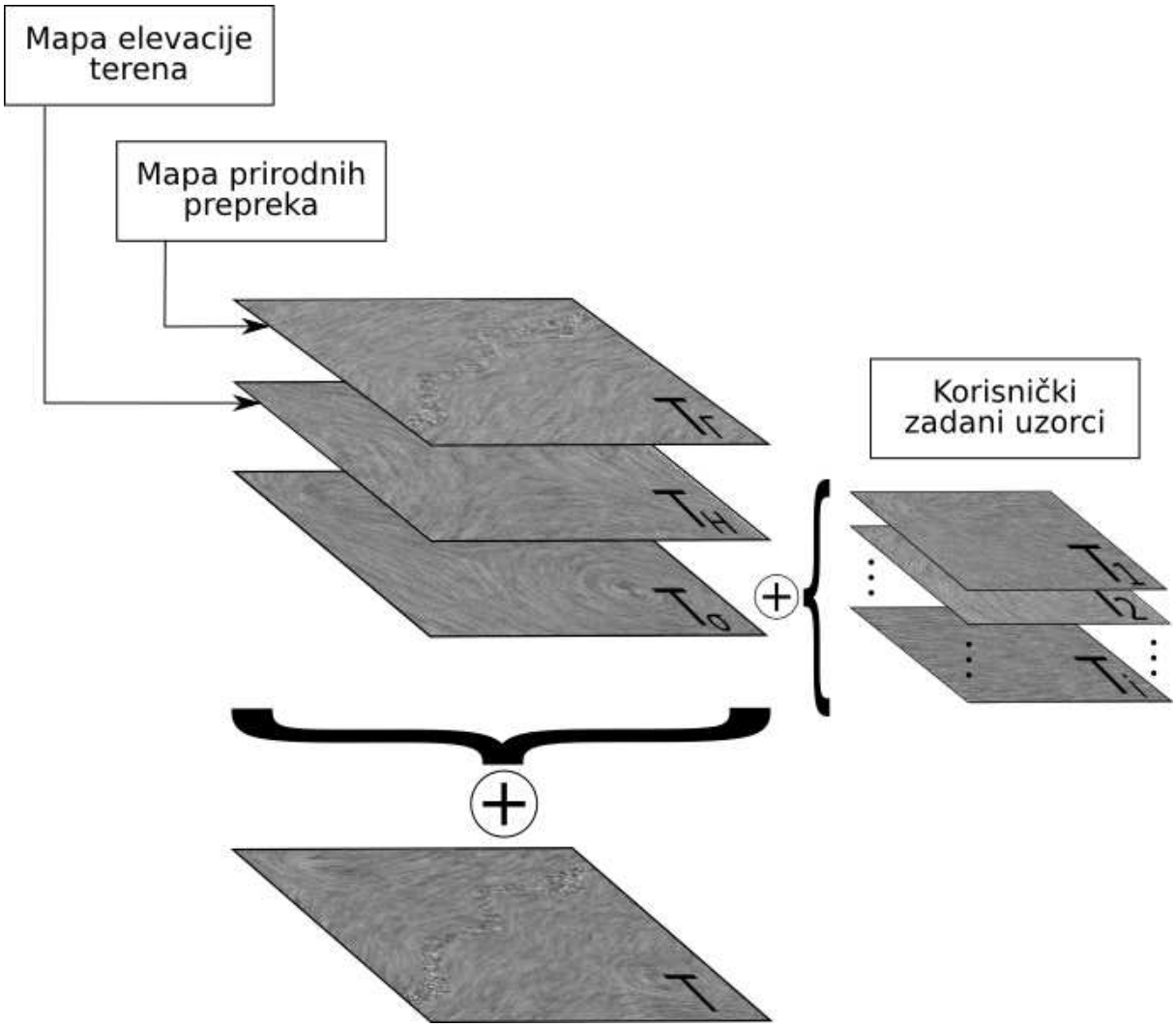
```
void FieldPainter::makePatterns()
{
    int lut[256];
    for (int i = 0; i < 256; i++)
    {
        lut[i] = i < 127 ? 0 : 255;
    }

    int phase[NPN][NPN];
    for (int i = 0; i < NPN; i++)
    {
        for (int j = 0; j < NPN; j++)
        {
            phase[i][j] = rand() % 256;
        }

    for (int k = 0; k < NPAT; k++)
    {
        int t = k*256/NPAT;
        GLubyte pat[NPN][NPN][4];

        for (int i = 0; i < NPN; i++)
        {
            for (int j = 0; j < NPN; j++)
            {
                pat[i][j][0] =
                pat[i][j][1] =
                pat[i][j][2] = lut[(t + phase[i][j]) % 255];
                pat[i][j][3] = alpha;
            }
        }

        glNewList(k, GL_COMPILE);
        glTexImage2D(
            GL_TEXTURE_2D, 0, 4, NPN, NPN, 0,
            GL_RGBA, GL_UNSIGNED_BYTE, pat);
        glEndList();
    }
}
```

3 Izgradnja tenzorskog polja

Rezultantno tenzorsko polje kreira se na temelju topografskih podataka (granica voda, šuma, parkova, etc.), visinske mape, statističkih podataka (gustoće naseljeništva), te korisnički zadanih ograničenja (poželjnih prometnih uzoraka).

Korisnik kao ulaz u sustav specificira topografske i statističke podatke u obliku 2D mapa. Mape mogu biti ručno nacrtane, proceduralno generirane ili dobivene na temelju stvarnih podataka. Koristeći elemente tenzorskog polja korisnik zadaje položaj i oblik poželjnih prometnih uzoraka. Za svaki ulaz u sustav kreira se tenzorsko polje koje odgovara ulaznim podacima. Rezultantno tenzorsko polje dobije se težinskim sumiranjem svih ulaznih tenzorskih polja.

Slika 10 shematski prikazuje proces generiranja tenzorskog polja na temelju svih ulaznih podataka. Na temelju korisnički zadanih elemenata polja stvara se tenzorsko polje T_σ, koje zajedno sa tenzorskim poljem stvorenim na temelju mape prirodnih prepreka T_Γ i tenzorskim poljem mape elevacije terena T_H, tvori rezultantno tenzorsko polje T koje navodi proces generacije prometne mreže.



Slika 10: Izgradnja tenzorskog polja

3.1 Prirodne granice

Prirodne granice nekog područja najviše utječu na izgled prometne mreže izgrađene na tom području. Obale mora, rijeka i jezera, granice šuma i parkova predstavljaju višemanje nepremostive prepreke za prometnu infrastrukturu. Potrebno je oblikovati tenzorsko polje koje poštuje tu činjenicu, te pokušava navesti proces stvaranja prometne mreže da obilazi granice prirodnih prepreka.

Za stvaranje tenzorskog polja koje poštuje prirodne granice terena (rijeke, jezera, šume, etc.) koristi se slijedeći postupak:

1. Iz zadane mape prirodnih prepreka (slika 11a) algoritmom detekcije granica [9] pronalaze se sve linije razgraničenja područja prirodnih prepreka s ostatkom mape (slika 11b).

2. Na temelju svake linije razgraničenja računa se aproksimacija te linije u obliku polilinije $L = \left\{\overline{v^k v^{k+1}} | k = 0, ..., n\right\}$ – linije sastavljene od n međusobno spojenih segmenata s vrhovima u v^k (slika 11c).

3. Za svaki segment $\overline{v^k v^{k+1}} \epsilon L$ definira se pripadajuće tenzorsko polje T_k čiji je glavni svojstveni vektor orijentiran u smjeru k-tog segmenta polilinije (slika 11c):

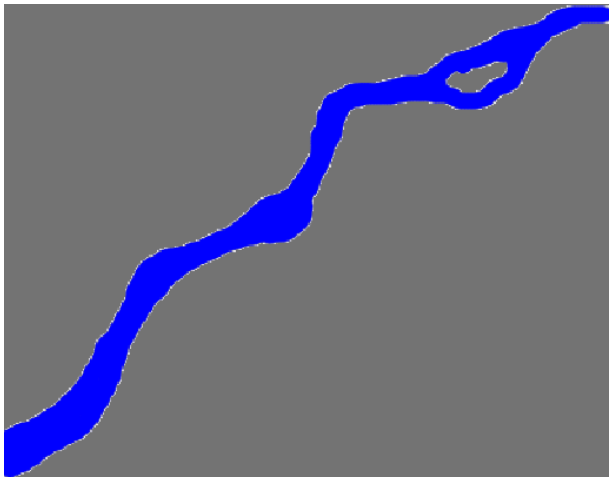
$$T_k(\mathbf{p}) = \mu \begin{bmatrix} \cos 2\theta & \sin 2\theta \\ \sin 2\theta & -\cos 2\theta \end{bmatrix},$$

$$\mu = \sqrt{(v_x^k - v_x^{k+1})^2 + (v_y^k - v_y^{k+1})^2},$$

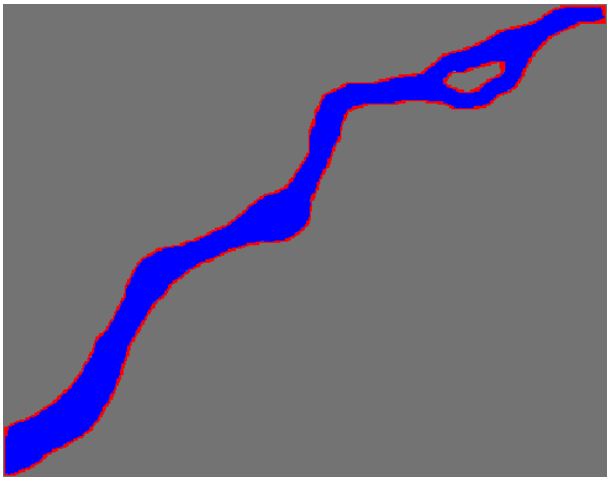
$$\theta = \arctan \frac{v_y^k - v_y^{k+1}}{v_x^k - v_x^{k+1}},$$

4. Tenzorsko polje koje poštuje prirodne granice (slika 11d) dobije se sumiranjem tenzorskih polja pojedinačnih segmenata po slijedećoj formuli:

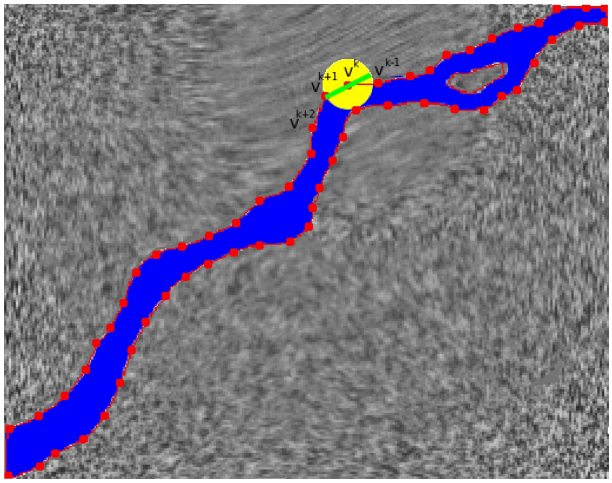
$$T_\Gamma(\mathbf{p}) = \sum_k e^{-d\|p - v^k\|} T_k(\mathbf{p}).$$



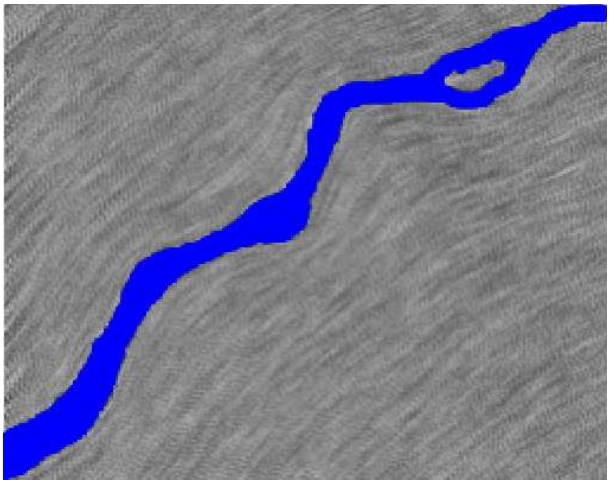
(a) Ulazna mapa



(b) Linije razgraničenja



(c) Aproksimacija polilinijom i tenzor za jedan segment



(d) Rezultantno polje

Slika 11: Stvaranje tenzorskog polja koje poštuje prirodne prepreke

Detekcija granica

Za male slike postupak može biti vrlo jednostavan: za svaku regiju dovoljno je pronaći jedan granični piksel te pratiti granicu regije u određenom smjeru (u smjeru kazaljke na satu ili obrnuto) dok se ne vratimo na početni pixel. Ova jednostavna tehnika nije pogodna za velike slike koje ne stanu cijele u glavnu memoriju računala, što uzrokuje velik pad performansi zbog pristupa sekundarnoj memoriji na disku. Čak i ako cijelokupna slika stane u glavnu memoriju takav u suštini nasumičan pristup pikselima uzrokovati će lošu iskorištenost *cache* memorije glavnog procesora (*cache trashing*).

Shapiro i Stockman opisuju algoritam nazvan *border* koji pronalazi granice svih regija u samo jednom prolazu kroz sliku [9]. Ulaz u algoritam *border* jer slika u kojoj su sve regije prethodno označene (labelirane), tj. svi pikseli iste regije su na neki način označeni istom oznakom. Isto tako svi pikseli koji ne pripadaju niti jednoj regiji (pozadinski pikseli) nose vlastitu oznaku. Izlaz iz algoritma je lista graničnih piksela, poredanih obrnuto od smjera kretanja kazaljke na satu, za svaku regiju iz ulazne slike.

Označavanje piksela generalno se obavlja nekim od algoritama segmentacije slike (engl. *image segmentation*), kao što je algoritam rasta regija (engl. *region growing*) [21]. U

našem slučaju mapa prirodnih prepreka već je implicitno segmentirana: svi pikseli RGBA vrijednosti 0 pripadaju pozadini, a svi ostali pripadaju regiji označenoj kao "prirodna prepreka".

Algoritam

Algoritam prolazi kroz ulaznu sliku skenirajući individualne piksele sa lijeva na desno, odozgo prema dolje, pritom skupljajući granične piksele u povezane nizove. Nizovi graničnih piksela tvore spojene segmente granica regija.

U svakom trenutku algoritam sadrži skup regija čije granice nisu u potpunosti još povezane i skup regija čije su granice u potpunosti pronađene. Za svaku regiju algoritam vodi povezanu listu segmenata koji su do tog trenutka formirani za tu regiju. Svaki segment je povezana lista koordinata graničnih piksela koja može rasti na oba kraja.

Ispis 2 prikazuje algoritam *border* u pseudokodu, dok slika 12 prikazuje primjer djelovanja algoritma na zadanu ulaznu sliku.

$y\|x$	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	0
2	0	1	1	1	0	0	0	0
3	0	1	1	1	3	3	3	0
4	0	1	1	1	3	3	3	0
5	0	0	0	0	0	3	3	0
6	0	2	2	2	2	3	3	0
7	0	2	2	2	2	3	3	0
8	0	0	0	0	0	3	3	0

(a) Ulazna slika

Labela	Lista (x,y)
1	(2,2), (3,2), (4,2), (4,3), (4,4), (3,4), (2,4), (2,3)
2	(2,6), (3,6), (4,6), (5,6), (5,7), (4,7), (3,7), (2,7)
3	(5,3), (5,4), (7,3), (7,4), (7,5), (7,6), (7,7), (7,8), (6,8), (6,7), (6,6), (6,5), (5,4)

(b) Izlaz iz algoritma

Slika 12: Algoritam *border*

```
procedure border(image)
begin
    regions := {}

    for y := 0 to image.height do
        for x := 0 to image.width do
            labela := image[x,y]

            if not regions.contains(labela) then
                regions[labela] = new Region;
            end

            neighbours := getNeightbours(image, (x,y), labela)

            if isBorderPixel(image, (x,y), neighbours) then
                added := 0

                for (xn,yn) in neighbours do
                    for segment in regions[labela] do
                        if (xn,yn) = segment.last then
                            segment.append((x,y))
                            added := 1
                            break
                        end
                    end
                end

                if not added then
                    regions[labela].add(new Segment((x,y)))
                end
            end
        end

        for region in regions do
            connectSegments(region)
        end
    end
end
```

Ispis 2: *Border* algoritam

3.2 Visinsko polje

Za stvaranje tenzorskog polja koje poštuje informaciju o elevaciji terena koristi se slijedeći postupak:

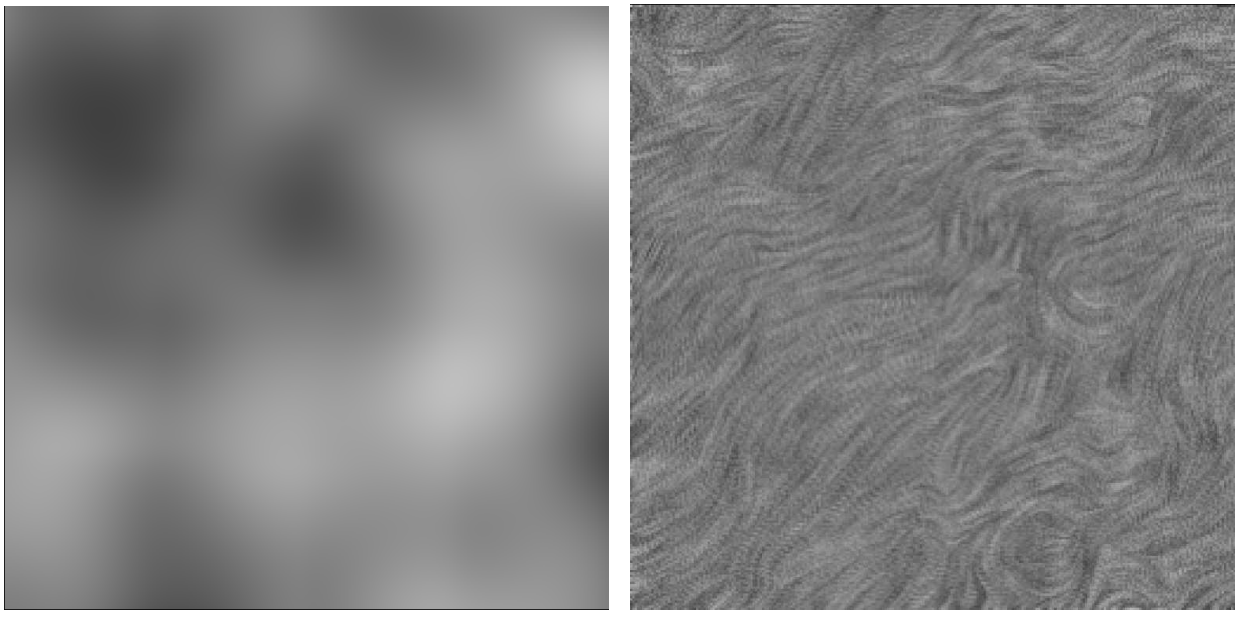
1. Na temelju visinske mape računa se gradijentna mapa po formuli: $\nabla H = (\frac{\partial H}{\partial x}, \frac{\partial H}{\partial y})$
2. Konstruira se tenzorsko polje čiji sporedni svojstveni vektor odgovara gradijentu visinskog polja:

$$T_H(x, y) = \mu \begin{bmatrix} \cos 2\theta & \sin 2\theta \\ \sin 2\theta & -\cos 2\theta \end{bmatrix},$$

$$\mu = \sqrt{\left(\frac{\partial H}{\partial x}\right)^2 + \left(\frac{\partial H}{\partial y}\right)^2},$$

$$\theta = \arctan \frac{\frac{\partial H}{\partial y}}{\frac{\partial H}{\partial x}} + \frac{\pi}{2}.$$

Slika 13 prikazuje primjer jedne visinke mape i tenzorskog nastalog na temelju opisanog postupka.



(a) Ulazna mapa (b) Rezultantno tenzorsko polje

Slika 13: Tenzorsko polje stvoreno na temelju visinske mape

3.3 Korisnički zadani uzorci

Korisniku se pruža mogućnost direktnog uređivanja tenzorskog polja korištenjem elemenata tenzorskog polja. Korištene tehnike dizajna tenzorskih polja opisali su Zhang et al. u svom radu *Interactive Tensor Field Design and Visualization on Surfaces*. Svaki element specificira jedan uzorak na zadanoj lokaciji. Različiti tipovi elemenata stvaraju različite uzorke:

- *Regularni element* $A(\mathbf{p}) = \mu \begin{bmatrix} \cos 2\theta & \sin 2\theta \\ \sin 2\theta & -\cos 2\theta \end{bmatrix}, \mu = \sqrt{v_x^2 + v_y^2}, \theta = \arctan \frac{v_y}{v_x}$, generira "rešetkasti" prometni uzorak. Rešetkasta prometna struktura je najvažniji prometni uzorak u većini gradova. Kod takve prometne strukture građevinske parcele nastaju iz dva ortogonalna seta usporednih prometnica. Slika 14a prikazuje tenzorsko polje stvoreno regularnim elementom sa $\theta = 45°$.

- *Centar* $A(\mathbf{p}) = \begin{bmatrix} y^2 - x^2 & -2xy \\ -2xy & -(y^2 - x^2) \end{bmatrix}, x = x_p - x_0, y = y_p - y_0$, generira radijalni uzorak sa centrom u točki $\mathbf{p}_o = (x_0, y_0)$. Radijalni uzorci u prometnim mrežama pojavljuju se u nekoliko različitih konteksta, od rezidencijalnih četvrti do čitavih gradskih centara (slika 14b).

- *Izvor* (engl. *node*) $A(\mathbf{p}) = \begin{bmatrix} x^2 - y^2 & 2xy \\ 2xy & -(x^2 - y^2) \end{bmatrix}, x = x_p - x_0, y = y_p - y_0$, generira izvor s centrom u točki $\mathbf{p}_o = (x_0, y_0)$. Izvor je vrlo sličan centru, s razlikom da su glavni i sporedni pravci zamijenili uloge. Centar se može koristiti u točkama lokalnog maksimuma mape populacije (slika 14c).

- *Klin* (engl. *wedge*) $A(\mathbf{p}) = \begin{bmatrix} x & y \\ y & -x \end{bmatrix}, x = x_p - x_0, y = y_p - y_0$, stvara klinasti uzorak sa vrhom u točki $\mathbf{p}_o = (x_0, y_0)$ (slika 14d).

- *Trisektor* $A(\mathbf{p}) = \begin{bmatrix} x & -y \\ -y & -x \end{bmatrix}, x = x_p - x_0, y = y_p - y_0$, uzrokovati će trodjelnu zvijezdu sa centrom u točki $\mathbf{p}_o = (x_0, y_0)$ (slika 14e).

- *Sedlo* (engl. *saddle*) $A(\mathbf{p}) = \begin{bmatrix} x^2 - y^2 & -2xy \\ -2xy & -(x^2 - y^2) \end{bmatrix}, x = x_p - x_0, y = y_p - y_0$, generira karakterističnu sedlasti uzorak sa centrom u točki $p_o = (x_0, y_0)$ (slika 14f).

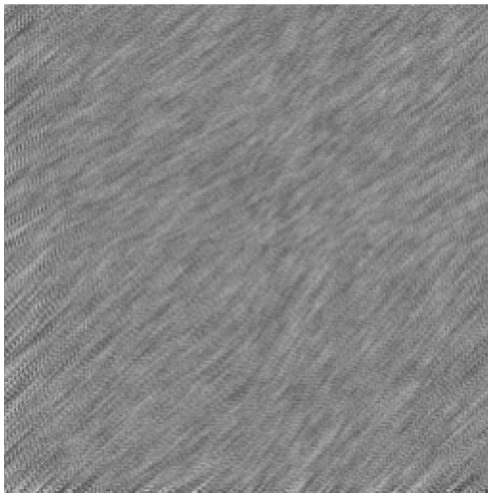
Svaki element tenzorskog polja pretvara se u *bazno* tenzorsko polje definirano na cijeloj domeni polja po slijedećoj formuli:

$$T(\mathbf{p}) = e^{-d\|p-p_0\|} A(\mathbf{p})$$

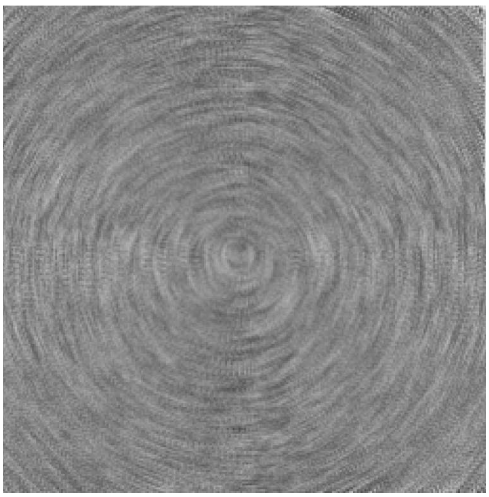
gdje je d konstanta opadanja (*delay*) radijalne bazne funkcije, $\mathbf{p}$ je zadana točka u domeni polja, a $\mathbf{p}_0$ je točka u kojoj je građevni element pozicioniran [2]. Rezultantno tenzorsko polje dobije se težinskim sumiranjem svih baznih polja:

$$T_\sigma(\mathbf{p}) = \sum_i w_i T_i(\mathbf{p})$$

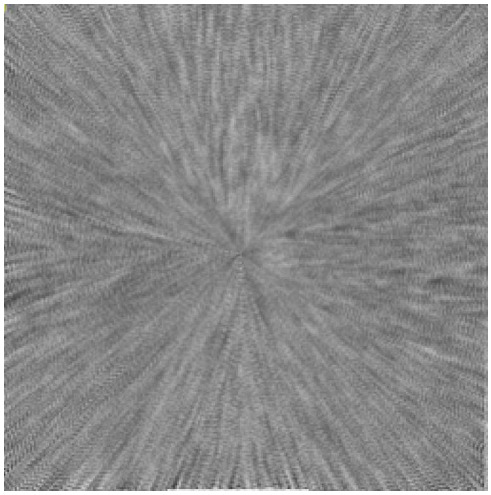
gdje je T_i bazno tenzorsko polje koje odgovara i-tom građevnom elementu polja, a $w_i \epsilon [0, 1]$ je zadani težinski koeficijent i-tog baznog polja.



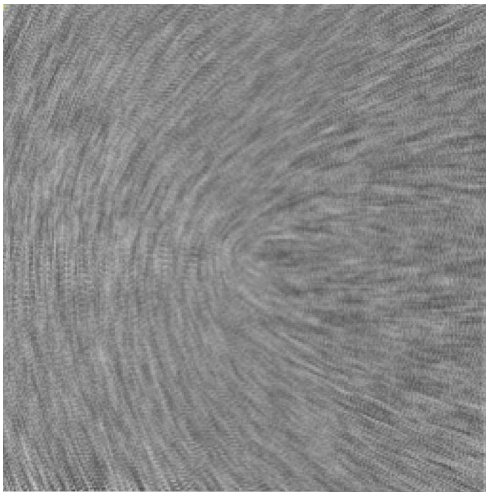
(a) Regularni element



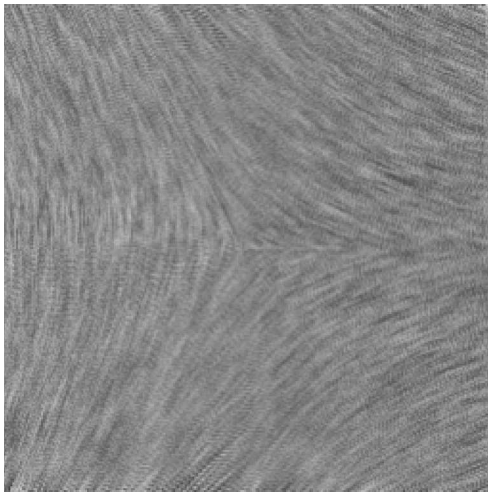
(b) Centar



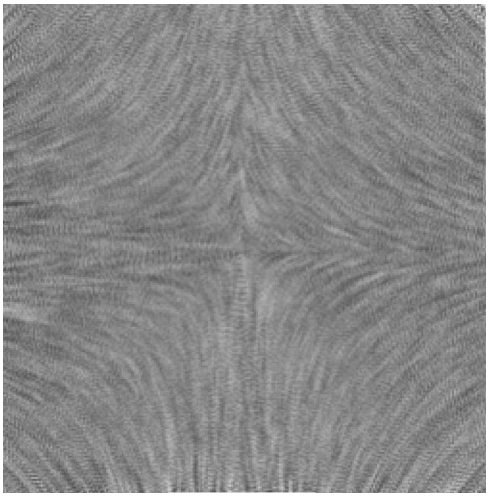
(c) Izvor



(d) Klin



(e) Trisektor



(f) Sedlo

Slika 14: Elementi tenzorskog polja

4 Izgradnja prometne mreže

Prometna mreža generira se na temelju prethodno izgrađenog tenzorskog polja. Postupak izgradnje prometne mreže bazira se na praćenju glavnih i sporednih hiperlinija toka.

Važan aspekt svih uzoraka prometnih mreža je postojanje dva dominantna smjera koja proizlaze iz potrebe za što boljim iskorištenjem prostora. Iz tenzorskog polja proizlaze dva seta linija toka – jedan prati polje glavnog svojstvenog vektora dok drugi prati polje sporednog svojstvenog vektora [2].

4.1 Praćenje linija toka

Praćenje određene hiperlinije toka svodi se na integriranje vektorskog polja glavnog ili sporednog svojstvenog vektora tenzora T, počevši od točke p_0:

$$p = p(l), \quad l\epsilon[0, +\infty\rangle$$

$$p(0) = p_0 = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$$

$$E_v(p) = \frac{\partial p}{\partial l}$$

$$p(l_k + \Delta l) = p(l_k) + \int_{l=l_k}^{l=l_k+\Delta l} E\Big(p(l)\Big) dl$$

Za ravnomjerno polaganje mreže linija toka koristi se adaptacija algoritma za praćenje linija opisan u radu Jobarda i Lefera [13]. Za numeričku integraciju koristi se Runge-Kutta postupak:

$$p_{k+1} = p_k + h\frac{m_1 + 2m_2 + 2m_3 + m4}{6}$$

$$m_1 = E(p_k)$$

$$m_2 = E(p_k + h\frac{m_1}{2})$$

$$m_3 = E(p_k + h\frac{m_2}{2})$$

$$m_4 = E(p_k + hm_3)$$

Postupak praćenja hiperlinije se zaustavlja ukoliko se zadovolji jedan od uvijeta:

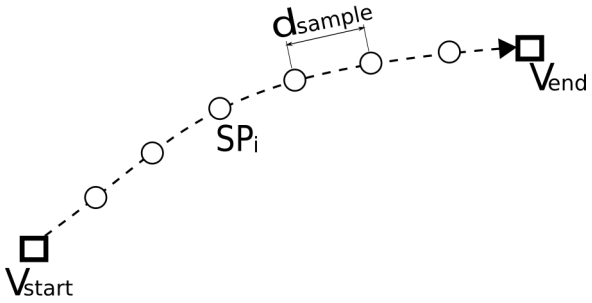
1. granica domene polja je dosegnuta,
2. dosegnuta je degenerirana točka,
3. praćena hiperlinija sječe postojeću hiperliniju,
4. dosegnuta je maksimalna specificirana duljina hiperlinije.

Ispraćena hiperlinija toka koja zadovoljava uvijet minimalne dužine postaje kandidat za novu prometnicu prometne mreže. U kontekstu izgradnje prometne mreže jedna prometnica naziva se *brid*, zbog uske povezanosti postupka izgradnje prometne mreže sa izgradnjom grafa povezanosti. Također zbog istog razloga mijesto gdje se sijeku prometnice – križanja, čvorišta – nazivaju se *vrhovi*.

4.2 Izgradnja bridova

Na temelju ispraćene hiperlinije toka stvaraju se bridovi prometne mreže. Jedan brid u prometnoj mreži definiraju slijedeći podaci:

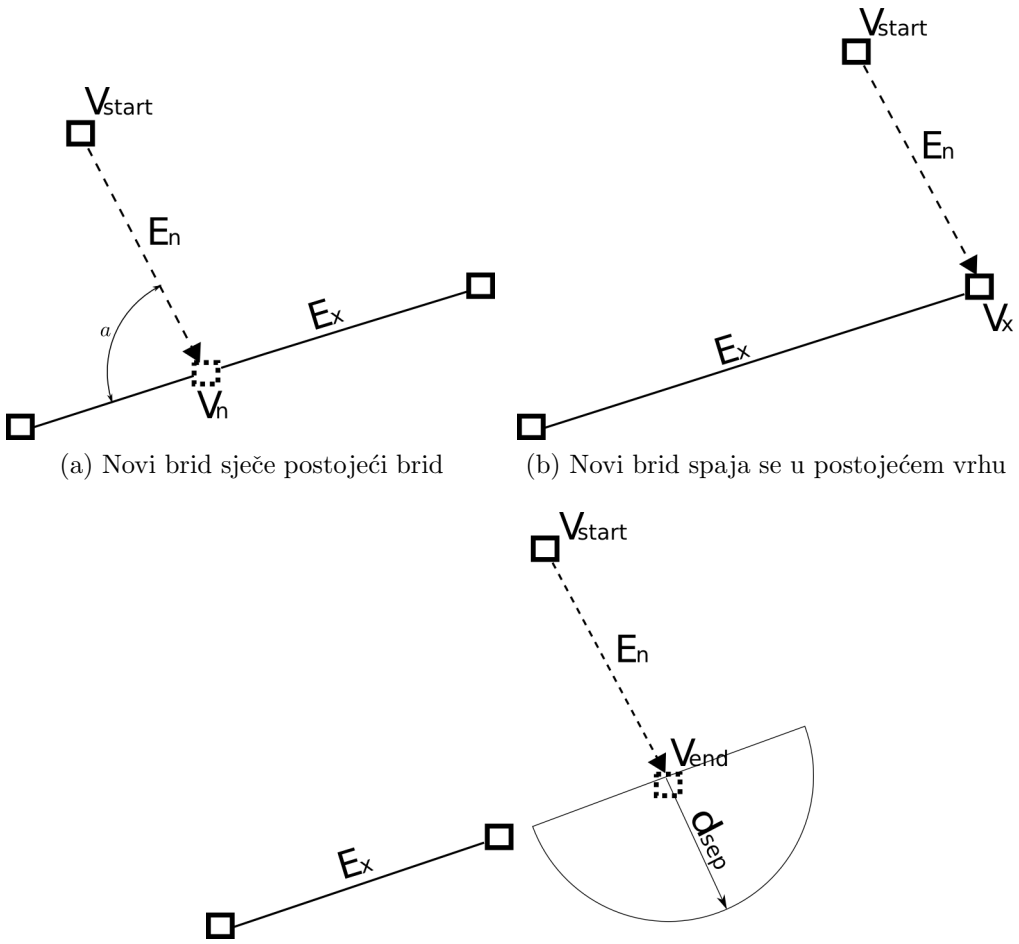
- *početna točka* V_{begin} – točka u kojoj brid zapičinje,
- *krajnja točka* V_{end} – točka u kojoj brid završava,
- *trag* (engl. *trace*) – poredani niz točaka SP_i koje se nalaze na pripadajućoj hiperliniji te su međusobno udaljene najviše d_{sample}.



Slika 15: Shematski prikaz brida prometne mreže

Da li će nova linija postati brid i u kakvom obliku ovisi o izgledu do tada izgrađene prometne mreže, tj. o položaju postojećih bridova.

- Ukoliko je krajnja točka novog brida E_n udaljena $d_{connect}$ od točke traga postojećeg brida E_x, te ako novi brid sa postojećim zatvara kut ne manji od 45°, tada se spojna točka bridova pretvara u novi vrh V_n, postojeći brid se sječe na dva dijela, a novi vrh V_n postaje završna točka novog brida (slika 16a).
- Ukoliko je krajnja točka novog brida E_n udaljena $d_{connect}$ od postojećeg vrha V_x, tada postojeći vrh postaje krajnja točka novog brida, te tako novi brid završava u postojećem prometnom čvorištu (slika 16b).
- Ukoliko novi brid ne sječe postojeći niti se spaja u postojeće čvorište tada se na poziciji završne točke V_{end} novog brida stvara novi vrh prometne mreže V_n, pod uvijetom da d_{sep} oko novog vrha nema postojećih vrhova (slika 16c).



(a) Novi brid sječe postojeći brid

(b) Novi brid spaja se u postojećem vrhu

(c) Novi brid stvara novi vrh

Slika 16: Stvaranje novog brida

4.3 Rasadne točke

Praćenje hiperlinije toka uvijek započinje u tzv. *rasadnoj* točki (engl. *seed point*). Početni skup rasadnih točkaka u tenzorskom polju može biti zadan od strane korisnika ili može biti proceduralno generiran.

Sve aktualne rasadne točke drže se u prioritetnoj listi. Prioritet svake točke može ovisiti o raznim parametrima, a kao najvažniji navode se [2]:

- udaljenost do najbliže prirodne prepreke d_b,
- udaljenost do najbliže degenerirane točke tenzorskog polja d_s,
- udaljenost do najbližeg centra populacije d_p.

Wonka et al. u [2] predlažu slijedeći izraz za računanje prioriteta:

$$\omega_{p_s} = e^{-d_b} + e^{-d_s} + e^{-d_p}$$

Postupak izgradnje prometne mreže u svakoj iteraciji iz prioritetne liste uzima točku najvišeg prioriteta. Praćenjem hiperlinije toka iz te točke potencionalno se stvara novi brid. U slučaju da je novi brid uspješno stvoren, na lokaciji krajnje točke novog brida V_{end} stvara se nova rasadna točka koja se dodaje u prioritetnu listu.

4.4 Prostorna segmentacija

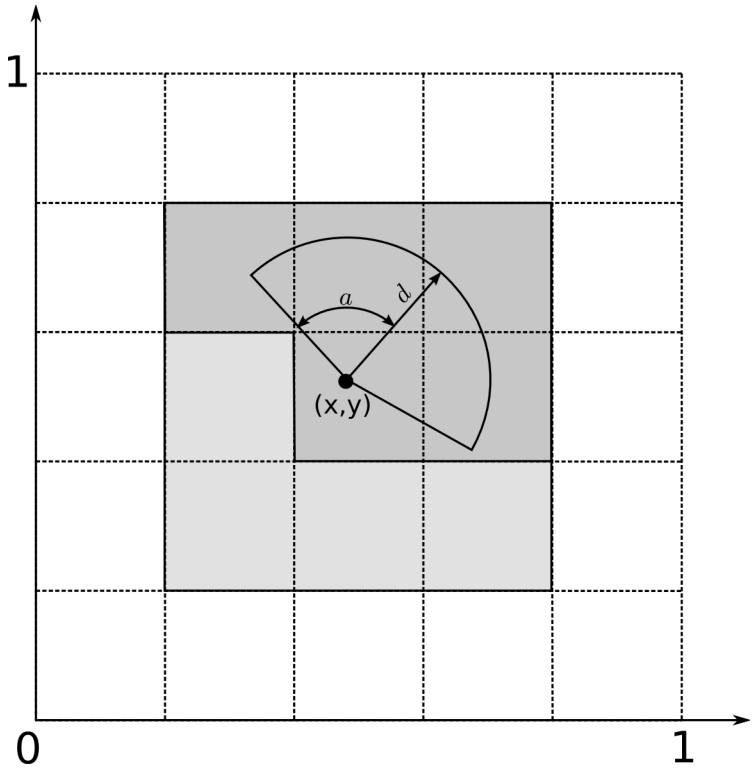
Operacija izračuna okoline pojedinih točaka u postupku praćenja jedne linije toka obavlja se veći broj puta – potrebno je detektirati prostorne bliske točke postojećih bridova, pronaći postojeće vrhove, itd. Imajući na umu da se u cijelokupnom postupku generacije prometne mreže prati veliki broj linija toka, može se zaključiti da bi ta operacija mogla lako postati usko grlo cijelog postupka. Kako bi se operacija što je više moguće ubrzala koristi se postupak prostorne segmentacije domene polja.

Domena polja S prostorno se segmentira pravokutnom mrežom G. Svaki segment te mreže $G_{i,j}$ sadrži listu točaka (vrhova V, točaka traga brida SP) koje prostorno pripadaju u taj segment. Tako kod izračuna okoline nije potrebno u obzir uzeti sve točke domene, već samo one koje pripadaju u isti segment u kojemu se nalazi točka čiju okolinu pretražujemo.

Za zadanu pravokutnu mrežu G sa M redaka i N stupaca, točka $\mathbf{p} = (x, y)$ pripada segmentu mreže $G_{i,j}$, gdje su:

$$i = min\big(\lfloor xM \rfloor, M - 1\big),$$
$$j = min\big(\lfloor yN \rfloor, N - 1\big).$$

U ispisu 3 pseudokodom je prikazana procedura za pretragu okoline zadane točke. Osim točke $p = (x,y)$ čiju okolinu pretražujemo, proceduri se zadaju vektor smjera d i kut pretrage a. Slika 17 ilustrira značenje pojedinih parametara procedure.



Slika 17: Pretraga okoline zadane točke

Tamno osjenčana područja predstavljaju segmente koji mogu sadržavati točke koje ulaze u područje pretrage, dok točke u svijetlo osjenčanim područjima ne bi trebale ući u pretragu, ali će zbog pojednostavljenja algoritma ipak biti uzete u obzir.

Parametrima d i a specificira se područje pretrage. Pretražuju se segmenti mreže koji sadrže točke koje potencijalno ulaze u područje pretrage. Neka točka se se nalazi u

zadanom području pretrage ako je od točke p udaljena ne više od duljine vektora d i ako sa njima zatvara kut ne veći od a.

```
procedure find(p=(x,y) : Vector, d : Vector, a : Real)
begin
    row := min(⌊x * G.rows⌋, G.rows − 1)
    col := min(⌊y * G.cols⌋, G.cols − 1)

    adjRows := ⌈|| d || *G.rows⌉
    adjCols := ⌈|| d || *G.cols⌉

    L := []

    for dr := −adjRows to adjRows do
        for dc := −adjCols to adjCols do
            r := row + dr
            c := col + dc

            if c < 0 or c >= G.cols or r < 0 or r >= G.rows do
                continue
            end

            for element in G[r,c] do
                dir := element.position − p
                dis := || dir ||

                if dis ≤ || d || then
                    if dis ≤ ϵ
                        L ← element
                    else
                        angle := arccos(dir/dis * d)

                        if angle < a then
                            L ← element
                        end
                    end
                end
            end
        end
    end

    return sortByDistance(L)
end
```

Ispis 3: Prostorna pretraga

Povećanjem gustoće mreže G smanjuje se prosiječan broj usporedaba koje se obavljaju, ali se povećava potrebna memorija za strukture podataka same mreže.

4.5 Graf povezanosti

Usporedno sa postupkom stvaranja prometne mreže obavlja se postupak izgradnje grafa povezanosti prometne mreže $G = (V, E)$, gdje je V skup vrhova grafa – točaka gdje se pojedine prometnice spajaju, a E je skup bridova – cestovnih segmenata koji se protežu između dva spojišta.

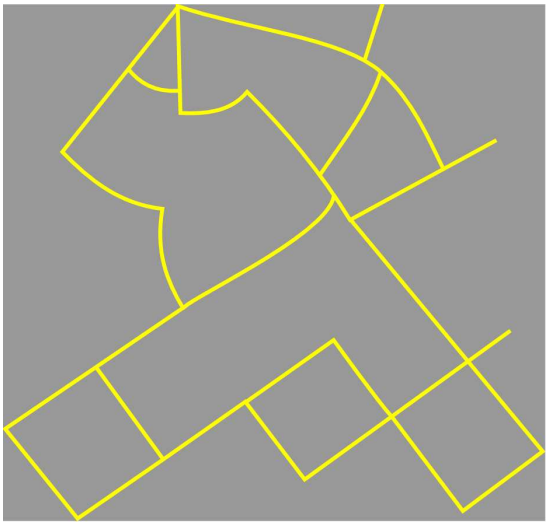
Prije početka procesa izgradnje prometne mreže graf G sadrži vrhove u točkama koje odgovaraju pozicijama inicijalnih rasadnih točaka. Svakom iteracijom postupka izgradnje u graf se dodaju vrhovi koji odgovaraju krajnjim točkama novo-stvorenih bridova V_{end}, te se u graf dodaje veza izmežu vrhova koji su u toj iteraciji postali spojeni prometnicom.

Tako izgraženi graf G koristi se u slijedećoj fazi postupka za pronalaženje kvartova i urbanih blokov unutar prometne mreže. Taj postupak obavlja se traženjem poligona u izvedenom grafu.

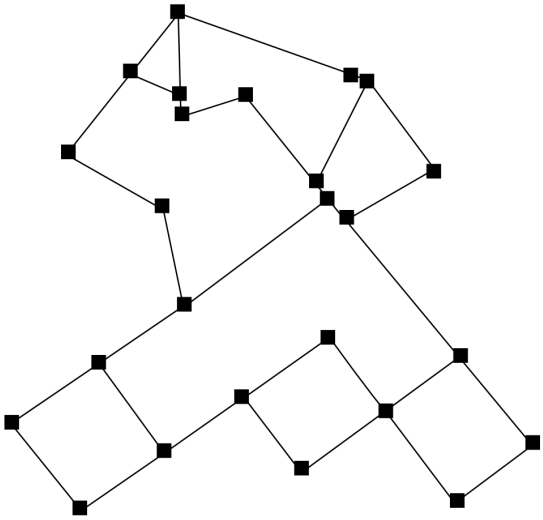
5 Formiranje urbanih područja

Izlaz iz faze generiranja prometne mreže (bilo glavne ili sporedne) u suštini je samo skup linija (bridova) međusobno spojenih u spojnim točkama (vrhovima). Takvu prometnu mrežu može se izravno reprezentirati planarnim grafom $G = (V, E)$, gdje točke u kojima se linije spajaju tvore skup vrhova grafa $\mathcal{V}(G) = V$, a linije koje se protežu između spojnih točaka tvore skup bridova grafa $\mathcal{E}(G) = E$. Takav graf stvoren na temelju generirane prometne mreže nazivati ćemo *izvedeni graf*.

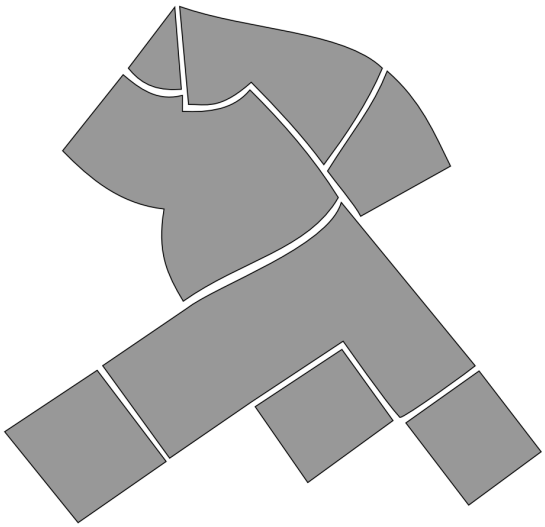
Izvedenim grafom koristiti ćemo se za pronalazak zatvorenih područja domene koja su sa svih strana omeđena prometnicama prometne mreže u kombinaciji sa granicama prirodnih prepreka. Područja omeđena prometnicama glavne prometne mreže nazivati ćemo *kvartovi* (engl. *districts*), a područja omeđena lokalnim kvartovskim ulicama nazivati ćemo *blokovi* (engl. *blocks*).



(a) Ulazna prometna mreža



(b) Izvedeni graf



(c) Izolirani individualni poligoni

Slika 18: Detekcija poligona iz prometne mreže

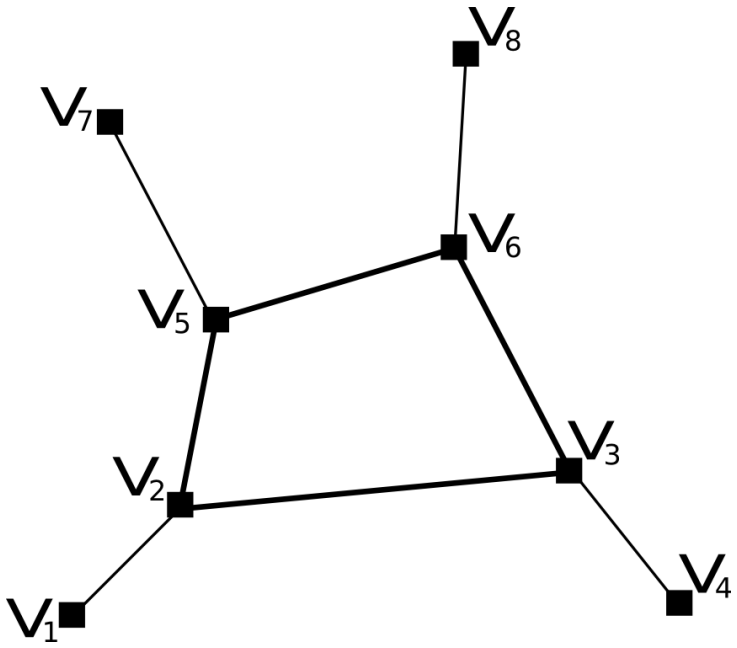
5.1 Matematičke definicije

U ovom odijeljku navesti ćemo neke važnije definicije iz područja *teorije grafova* koje su nam potrebne za razumijevanje kasnije navedenih postupaka. Dobro poznavanje teorije grafova važno je za mnoga područja računalne znanosti, pa čitaoca upućujem na dobru *on-line* knjigu *Graph Theory with Applications* [18].

Planarnim grafom naziva se graf G koji je moguće nacrtati u ravnini na način da se bridovi dodiruju samo u vrhovima. Slika 19 ilustrira primjer planarnog grafa sa 8 vrhova i 8 bridova.

Putem (engl. *path*) na grafu G naziva se niz vrhova grafa $[v_1, v_2, ..., v_n]; v_i \in \mathcal{V}(G)$ takav da su svi $(v_1, v_2), (v_2, v_3), ..., (v_{n-1}, v_n)$ bridovi grafa G te da su svi v_i u nizu različiti.

Ciklus u grafu G je podskup skupa bridova $C \subseteq \mathcal{E}(G)$ koji tvori trag specifičan po tome da prvi vrh u nizu odgovara zadnjemu. Na slici 19 debljim linijama označen je ciklus u prikazanom grafu.



Slika 19: Primjer planarnog grafa sa jednim ciklusom.
Bridovi [(2,3), (3,6), (6,5), (5,2)] tvore ciklus duljine 4.

Svakom ciklusu C pridružen je *vektor incidencije* (engl. *incidence vector*) $\mathbf{x} = [x_e]$ indeksiran na skupu bridova grafa:

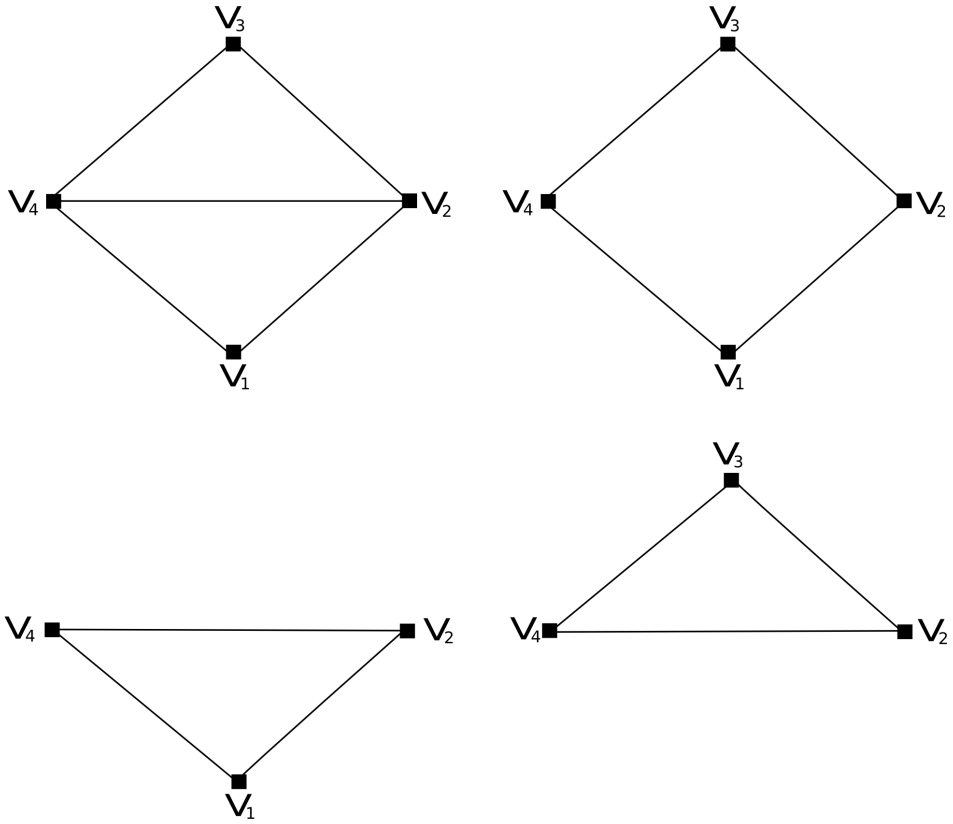
$$x_e = \begin{cases} 1 & e \in C \\ 0 & e \notin C \end{cases}$$

$$e \in \mathcal{E}(G)$$

Ciklusnim prostorom naziva se vektorski prostor nad konačnim poljem $\mathbb{F}_2$ generiran vektorima incidencije ciklusa [15].

Skup ciklusa $C \subseteq \mathcal{E}(G)$ naziva se *ciklusna baza* grafa G ako tvori bazu ciklusnog prostora, tj. C se sastoji isključivo od elementarnih ciklusa [14]. Na slici 20 prikazan je graf sa tri ciklusa čije međusobne kombinacije tvore tri ciklusne baze.

Ciklusna baza u kojoj je zbroj težina ciklusa minimalan naziva se *minimalna ciklusna baza* (MCB) [15]. Na slici 20 ciklusna baza $\{[v_1, v_2 v_4], [v_2, v_3, v_4]\}$ je minimalna.



Slika 20: Primjer planarnog grafa sa tri ciklusa.
Prikazani graf (gore lijevo) sadrži tri ciklusa: $[v_1, v_2, v_3, v_4]$
(gore desno), $[v_1, v_2 v_4]$ (dolje lijevo), $[v_2, v_3, v_4]$ (dolje desno).

Notacijom $\langle \mathbf{a}, \mathbf{b} \rangle$ označavamo standarni unutarnji umnožak (engl. *inner product*) vektora $\mathbf{a}$ i $\mathbf{b}$. Kažemo da su $\mathbf{a}$ i $\mathbf{b}$ *ortogonalni* ako $\langle \mathbf{a}, \mathbf{b} \rangle = 0$. Budući da se sve operacije obavljaju u polju $\mathbb{F}_2$, vrijediti će da je $\langle \mathbf{a}, \mathbf{b} \rangle = 1$ ako i samo ako vektor incidencije $\mathbf{a}$ sadrži neparan broj bridova iz $\mathbf{b}$ [15].

5.2 Postupak

Zbog prirode samog postupka generacije prometne mreže tako izvedeni graf G ima svojstvo *planarnosti*. To svojstvo omogućuje nam detekciju zatvorenih područja (poligona) unutar prometne mreže korištenjem postupka traženja *minimalne ciklusne baze* (MCB) u izvedenom grafu.

Algoritam za pronalazak svih ciklusa u grafu linearne vremenske složenosti prvi je objavio Maciej M. Syslo u radu "An Efficient Cycle Vector Space Algorithm for Listing all Cycles of a Planar Graph". Vremenska složenost algoritma je $O(VC)$, gdje je V broj vrhova grafa a C broj ciklusa u grafu.

Veliki problem je što broj ciklusa u planarnom grafu raste eksponencijalno sa brojem vrhova [14]. Zbog te činjenice nije praktično pronaći sve poligone koji se mogu mogu konstruirati iz bridova grafa. Ferreira et al. [14] opisuju postupak pronalaženja minimalnih poligona – onih koji sadrže minimalan broj bridova i ne mogu se konstruirati spajanjem ostalih minimalnih poligona. Budući da je izvedeni graf planaran, to se svodi na traženje minimalne ciklusne baze grafa.

Traženje minimalne ciklusne baze grafa

Algoritam koji pronalazi MCB u polinomnom vremenu prvi je objavio J.D. Horton u radu "A polynomial-time algorithm to find the shortest cycle basis of a graph" [17]. Vremenska složenost Hortonovog algoritma je $O(E^3V)$. Hortonov algoritam ne pronalazi cikluse po redu, nego prvo računa nadskup MCB-a, nazovimo ga *Hortonovim skupom* $\mathbb{HS}$, te nakon toga iz tog nadskupa izvlači MCB koristeći se Gaussovom eliminacijom. Dokaz da $\mathbb{HS}$ sadrži MCB dan je u [15]. Hortonov skup $\mathbb{HS}$ računa se koristeći slijedeći postupak:

- Za svaki vrh $w \in V$ i brid $e = (u, v) \in E$ kreira se ciklus kandidat $C = \mathrm{sp}(u, w) + \mathrm{sp}(w, v) + (u, v)$, gdje funkcija sp(a, b) vraća najkraći put između vrhova a i b;
- Ukoliko najkraći putevi $\mathrm{sp}(u, w)$ i $\mathrm{sp}(w, v)$ ne sadrže niti jedan zajednički vrh osim vrha w taj ciklus se dodaje u $\mathbb{HS}$, inače se ciklus odbacuje.

Izvlačenje MCB-a iz $\mathbb{HS}$ obavlja se koristeći Gaussovu eliminaciju:

- Prvo se svi ciklusi se sortiraju;
- i-ti ciklus iz Hortonova skupa je u MCB ako je linearno nezavisan od svih drugih koje smo odabrali iz prijašnjih $i - 1$ ciklusa;
- Proces se nastavlja dok se ne eliminiraju svi ciklusi.

U ispisu 4 pseudokodom je prikazan algoritam za pronalazak MCB-a na grafu G koji u [15] predlažu Dimitrios et al. za grafove bez zadanih težina bridova. U prvom dijelu procedure gradi se Hortonov skup $\mathbb{HS}$ koji sadrži MCB. U drugom dijelu procedure se računa MCB.

```
procedure mcb(G)
begin
    ℍ𝕊 := {}

    for w ∈ 𝒱(G) do
        for (u, v) ∈ ℰ(G) do
            if sp(u, w) ∩ sp(w, v) = {w} then
                C := sp(u, w) ∪ sp(w, v) ∪ (u, v)
                ℍ𝕊 ← C
            end
        end
    end

    sortByLength(HS)

    MCB := {}
    S_i := { e_i }

    for i := 1 to N do
        C_i := najkraci ciklus u ℍ𝕊 takav da ⟨C_i, S_i⟩ = 1
        MCB ← C_i
        for j := i+1 to N do
            if ⟨C_i, S_j⟩ = 1 then
                S_j := S_j + S_i
            end
        end
    end

    return MCB
end
```

Ispis 4: Traženje MCB-a

Najkraći putevi

Za rad algoritma za traženje MCB-a u grafu potrebna nam je informacija o najkraćim putevima između svih parova vrhova grafa.

Algoritam Floyda i Warshalla pronalazi najkraće puteve između svih vrhova zadanog grafa uspoređujući sve moguće puteve kroz graf između svakog para vrhova. Tu zadaću obavlja koristeći samo V^3 usporedba, što je značajno jer u grafu može biti do V^2 bridova a svaka kombinacija bridova mora biti testirana [22]. Algoritam Floyda i Warshalla je primjer *dinamičkog programiranja*, metode rješavanja kompleksnih problema njihovim razbijanjem na manje probleme.

Listing 5 prikazuje algoritam u pseudokodu. Procedura *allPairsShortestPaths* računa iznose duljina puteva između svih parova vrhova u grafu. Tako nakon izvršenja procedure polje *path[i,j]* sadržavati će iznos najkraće duljinu puta između vrhova *i* i *j*, dok će polje *next[i,j]* sadržavati informaciju o tome koji slijedeći vrh treba posjetiti ne bi li se najkraćim putem stiglo od vrha *i* do vrha *j*. Te dvije informacije koriste se u rekurzivnoj funkciji *getPath*, koja će na temelju vrijednosti polja *path* i *next* rekonstruirati najkraći put između vrhova zadanih parametrima funkcije.

```
procedure allPairsShortestPaths(G)
begin
    n := |V(G)| { broj vrhova u grafu G }

    path[n][n] = ∞
    next[n][n] = Ø

    for i := 1 to n do
        for j := 1 to n do
            if i = j then
                path[i,j] = 0
            else if (i,j) ∈ E(G) then
                path[i,j] = 1 { svi bridovi su iste tezine }
            else
                path[i,j] = ∞
            end
        end
    end

    for k := 1 to n do
        for i := 1 to n do
            for j := 1 to n do
                if path[i,k] + path[k,j] < path[i,j] then
                    path[i,j] := path[i,k] + path[k,j]
                    next[i,j] := k
                end
            end
        end
    end

    return (path, next)
end

procedure (path,next)::getPath(i,j)
begin
    if path[i,j] = ∞ then
        { ne postoji put izmedju zadanih vrhova }
        return NO_PATH
    end

    inter := next[i,j] { medjukorak }

    if inter = Ø then
        { izmedju zadanih vrhova postoji brid }
        return []
    else
        { rekonstruiraj puteve koristeci se medjukorakom }
        return getPath(i,inter) + [inter] + getPath(inter,j)
    end
end
```

Ispis 5: Algoritam Floyda i Warshalla

6 Rezultati

Tehnike i postupci opisani u prethodnim poglavljima i primjenjeni u sklopu ovog rada direktno su inspirirani radom Wonke et al. *Interactive Procedural Street Modeling*. U tome radu sažeto se navode sve korištene tehnike i prateća matematička podloga samih postupaka, no detalji o implementaciji su u potpunosti nepostojeći. Tako je glavna ideja oko koje se formirao ovaj rad zapravo pokušaj što korektnije implementacije navedenih tehnika kako bi rezultati što je više moguće vizualno nalikovali uzoru.

No, kako to već bude, dostići svoj uzor nije se pokazalo nimalo lagano. Tako su mnogi aspekti originala ostali nerealizirani. Nadam se da ovaj rad, ako ništa drugo, uspijeva pokazati prednosti prikazanih tehnika, među kojima se posebno ističe korištenja tenzorskih polja za modeliranje urbanih scena.

6.1 Utjecaj parametara

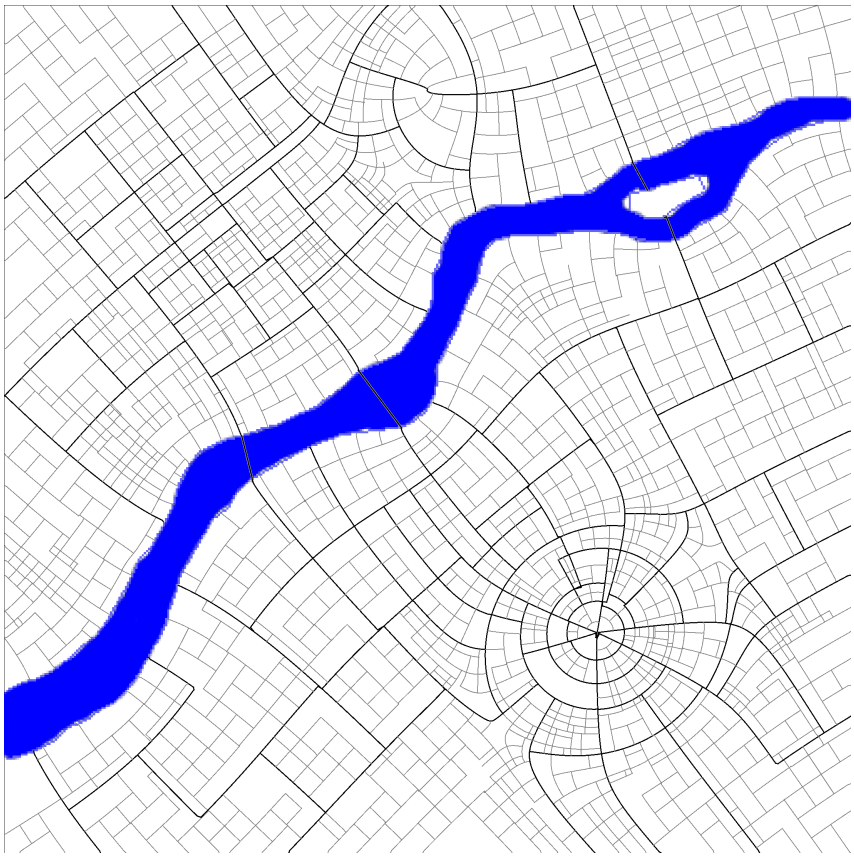
Izraženo obilježje proceduralnih tehnika je velika ovisnost krajnjeg rezultata o vrijednostima ulaznih parametara. Prikazani postupak generiranja prometne mreže ovisi o raznim parametrima kao što su konstante udaljenosti d_{sep} i $d_{connect}$, pozicije i brojnost početnih izvorišnih točaka p_s, položaji i parametri elemenata tenzorskog polja, težinski koeficijenti tenzorskog polja itd.

Na slici 21 prikazan je utjecaj položaja početnih točaka na krajnji rezultat postupka izgradnje prometne mreže. Slika 21b pokazuje rezultat postupka generiranja cijelokupne prometne mreže nakon što je u odnosu na prethodni slučaj, prikazan slikom 21a, promijenjena pozicija samo jedne početne izvorišne točke, i to u dijelu grada južno od rijeke. Vidljivo je da dio grada sjeverno od rijeke ima identični izgled glavne prometne mreže u oba slučaja, dok je glavna prometna mreža južnog dijela grada vidljivo različita. Naravno, općeniti izgled prometne mreže ostao je isti i nakon promjene pozicije početne točke, što i je smisao postupka proceduralnog modeliranja korištenjem tenzorskih polja.

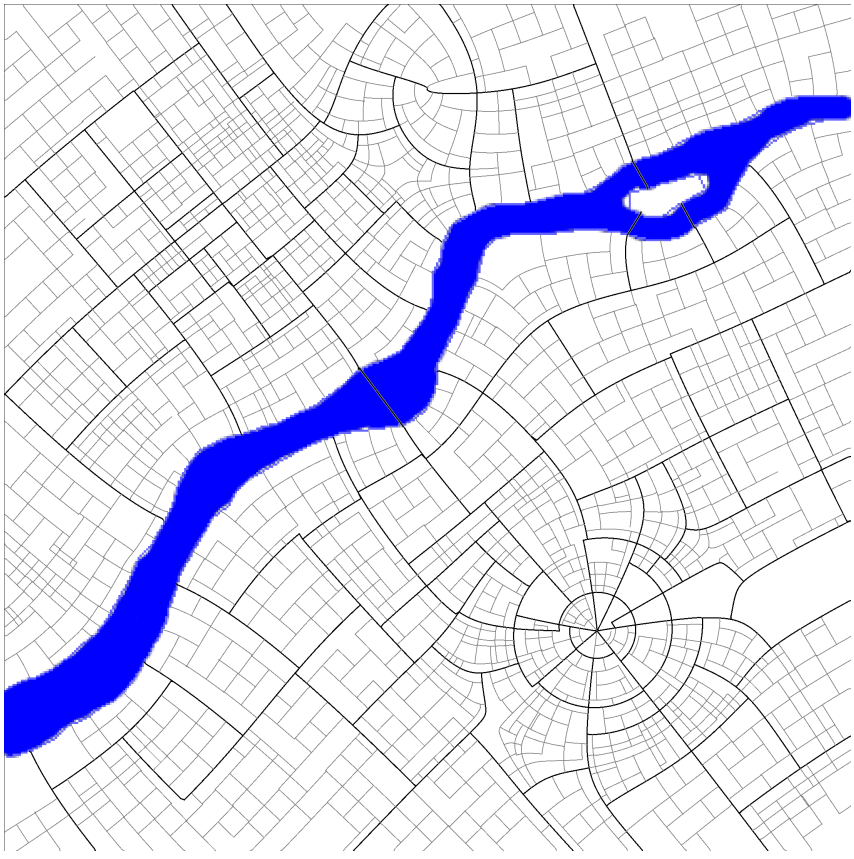
Izgled lokalnih prometnica razlikuje se u oba dijela grada što je rezultat različitog poretka zatvorenih područja (kvartova) pronađenih u glavnoj prometnoj mreži. Budući da se ukupni izgled glavne prometne mreže razlikuje između dva slučaja, tako se razlikuje i ulaz u fazu detekcije gradskih kvartova. Pronađeni kvartovi sjevernog dijela grada izgledom su jednaki u oba slučaja, no njihov poredak nije. Ta razlika ne bi bila prisutna kada bi lista kvartova dana kao ulaz fazi generiranja lokalnih prometnica bila po nekom ključu poredana. Ovaj primjer pokazuje kako proceduralne tehnike na izrazito suptilan način mogu ovisiti i o najmanjim promjenama parametara postupka.

Slika 22 pokazuje utjecaj parametra separacije prometnica d_{sep} na krajnji rezultat. Slika 22b prikazuje rijeđu prometnu mrežu u odnosu na onu prikazanu slikom 22a. Ta razlika nastala je kao rezultat povećanja vrijednosti parametra d_{sep} sa iznosa 0.6 na 0.8. Uloga parametra d_{sep} navedena je u poglavlju 4.2 i prikazana je na slici 16c.

Važno je još jednom primjetiti da jednako kao kod promjene položaja početnih točaka, tako i kod promjene parametra d_{sep} (a i ostalih parametara postupka) općeniti izgled krajnjeg rezultata ostaje jednak. Ta činjenica jasno pokazuje prednost korištenja tenzorskog polja za navođenje algoritma polaganja prometnica.

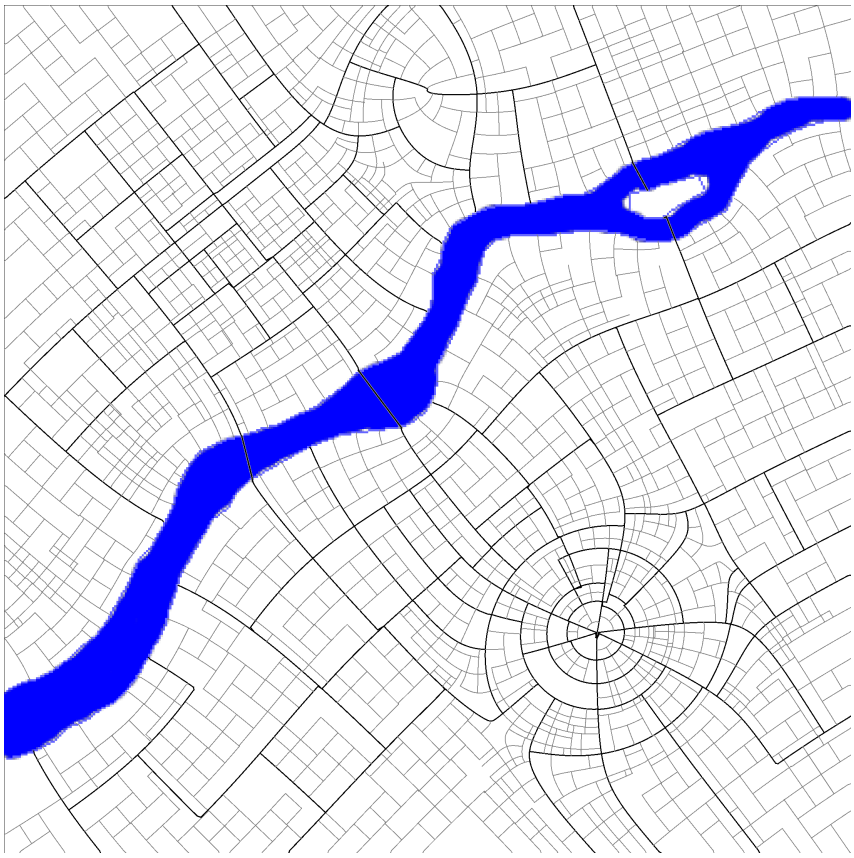


(a) Rezultat postupka prije promjene pozicije početne izvorišne točke

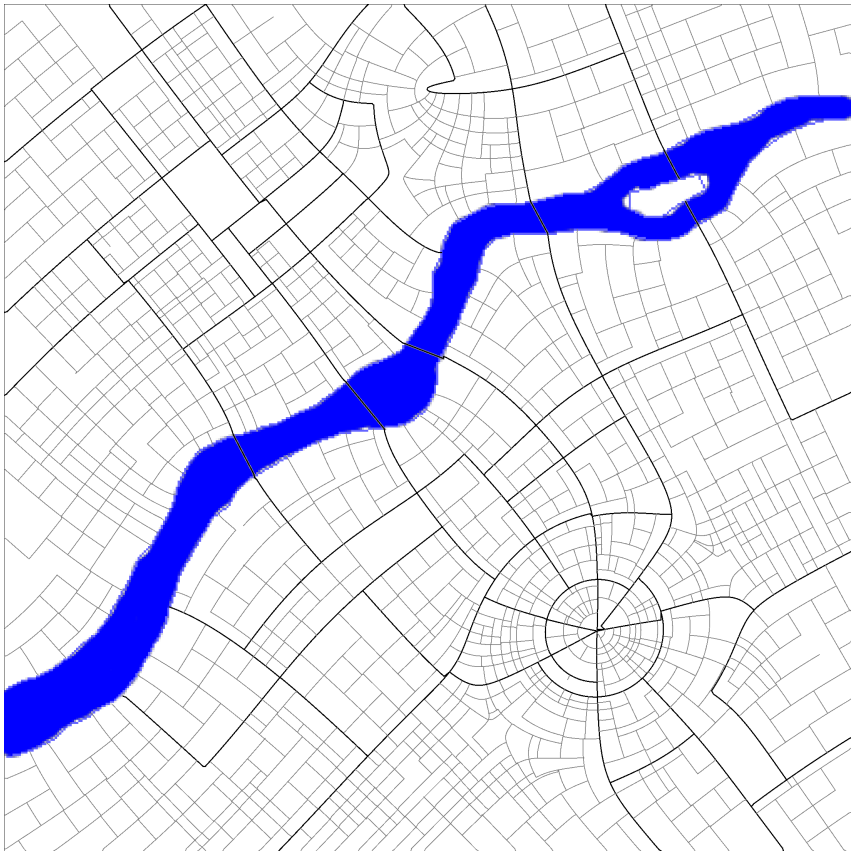


(b) Rezultat postupka nakon promjene pozicije početne izvorišne točke

Slika 21: Utjecaj pozicija početnih rasadnih točaka



(a) Rezultat postupka sa $d_{sep} = 0.6$



(b) Rezultat postupka sa $d_{sep} = 0.8$

Slika 22: Utjecaj parametra udaljenosti separacije

6.2 Vremenska složenost

Izražena ovisnost krajnjeg rezultata neke proceduralne tehnike o ulaznim podacima i parametrima samog postupka može ujedno biti i njezin najveći nedostatak. Upitna je iskoristivost tehnike na čiji rezultat je potrebno predugo čekati bez barem općenite ideje o njegovom izgledu.

Jedna od glavnih ideja postupka kojeg su u *Interactive Procedural Street Modeling* opisali Wonka et al. je integracija proceduralnog modeliranja sa korisničkom interakcijom [2]. Upravo zbog tog razloga naglasak je stavljen na interaktivnost samog postupka, tj. na interakciju korisnika sa samim proceduralnim procesom. Tako korisnik u svakom trenutku ima uvid u izgled krajnjeg rezultata. Kako bi taj pristup bio moguć sam proceduralni postupak mora se izvršavati dovoljno brzo – ako ne u potpunosti onda barem po svojim pojedinim fazama i ako ne u punom detalju onda barem dovoljno za približni prikaz rezultata.

Postupak naveden u prethodnim poglavljima može se podijeliti u zasebne faze, kao što je opisano u poglavlju 1.4. Faza generacije prometne mreže nalazi se na drugom mjestu po vremenskoj složenosti. Njena vremenska zahtjevnost najviše ovisi o rezoluciji praćenja hiperlinija toka i koraku numeričke integracije. Prosječno vrijeme izgradnje glavne prometne mreže kreće se oko 5 sekundi, dok je za cijelokupnu prometnu mrežu potrebno ne više od 40 sekundi. Za potrebe vizualizacije prometna mreža može se iscrtati smanjenom rezolucijom, što može znatno smanjiti potrebno vrijeme.

Vremenskom složenosti dominira postupak pronalaženja zatvorenih područja unutar prometne mreže koji se primjenjuje u fazama stvaranja kvartova i blokova. Tablica 1 navodi nekoliko primjera trajanja postupka.

Broj vrhova	Broj bridova	Pronađeno ciklusa	Trajanje (s)
112	168	67	4,3
160	244	93	12
168	257	100	14
182	272	101	18
186	288	108	21
185	291	115	22
190	303	117	22
199	320	130	27
216	351	138	52
226	362	147	56
241	406	169	84
297	489	199	163

Tablica 1: Trajanje postupka detekcije ciklusa

Broj vrhova i bridova zadanog izvedenog grafa ovisi o gustoći prometne mreže. Trajanje postupka ne iznenađuje ako se ima na umu da je vremenska složenost Hortonovog algoritma traženja minimalne ciklusne baze $O(E^3V)$. Olakotna okolnost ove činjenice leži u tome da je faza pronalaska pojedinih gradskih blokova na samom kraju cijelokupnog postupka.

6.3 Primjeri

U ovome poglavlju prikazati će se nekoliko primjera gradova generiranih postupkom opisanim u ovome radu. Svaki od navedenih primjera prikazuje određeni aspekt postupka na kojeg treba obratiti pažnju.

Primjer obalnog grada

Na slici 23 prikazan je primjer generiranja jednog fiktivnog grada na riječnom ušću. Slike 23a, 23b i 23c prikazuju ulazne podatke na temelju kojih se generira model grada prikazan slikom 23d. Može se primjetiti kako je ulaznom mapom prirodnih prepreka postignuto odvajanje gradskih granica od riječnog korita i obale mora. Tablica 2 prikazuje trajanje postupka generacije po pojedinim fazama.

Faza	Trajanje (ms)	Trajanje (%)
Generiranje glavne prometne mreže	5313	6,15
Kreiranje kvartova	22707	26,27
Generiranje lokalnih prometnica	24292	28,11
Kreiranje blokova	34116	39,47

Tablica 2: Trajanje postupka po fazama

Primjer grada na izrazito brdovitom terenu

Slikom 24 prikazano je generiranje fiktivnog grada na izraženo brdovitom terenu. Slika 24a prikazuje ulazne podatke koji se sastoje od kombinacije mapa elevacije terena i prirodnih prepreka. U ovome slučaju prirodna prepreka svodi se na jednu manju rijeku koja teče kroz grad. Ono što ovome gradu daje poseban oblik je izražena brdovitost terena na kojemu se grad generira. Slika 24b pokazuje rezultat u obliku skice prometne karte, dok slika 24c pokazuje rezultat u obliku 2D modela. Može se primjetiti kako neravan teren utječe na krajnji rezultat u vidu određene doze neuređenosti rezultantne prometne mreže.

Primjer rekonstrukcije postojećeg grada

Na slici 25 prikazan je rezultat modeliranja grada na temelju podataka o stvarnom gradu na toj lokaciji. U ovom slučaju radi se o Zagrebu. Naravno, generirani grad uopće ne nalikuje postojećemu što se i nije moglo očekivati. No i bez nekih velikih očekivanja pokazalo se da ovaj postupak nije prikladan za rekonstrukciju postojećih gradova.

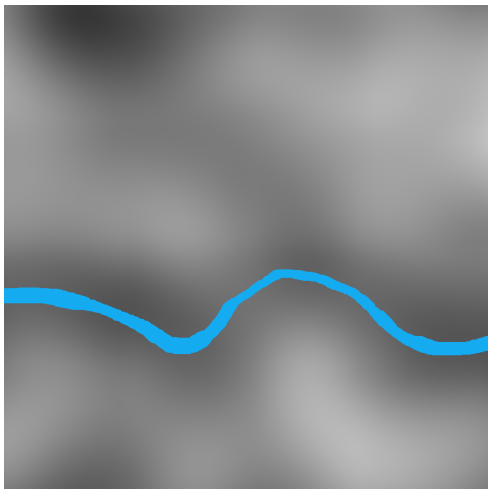
(a) Karta prirodnih prepreka

(b) Pozadinska slika

(c) Karta gustoće naseljenosti

(d) Rezultat

Slika 23: Primjer fiktivnog obalnog grada

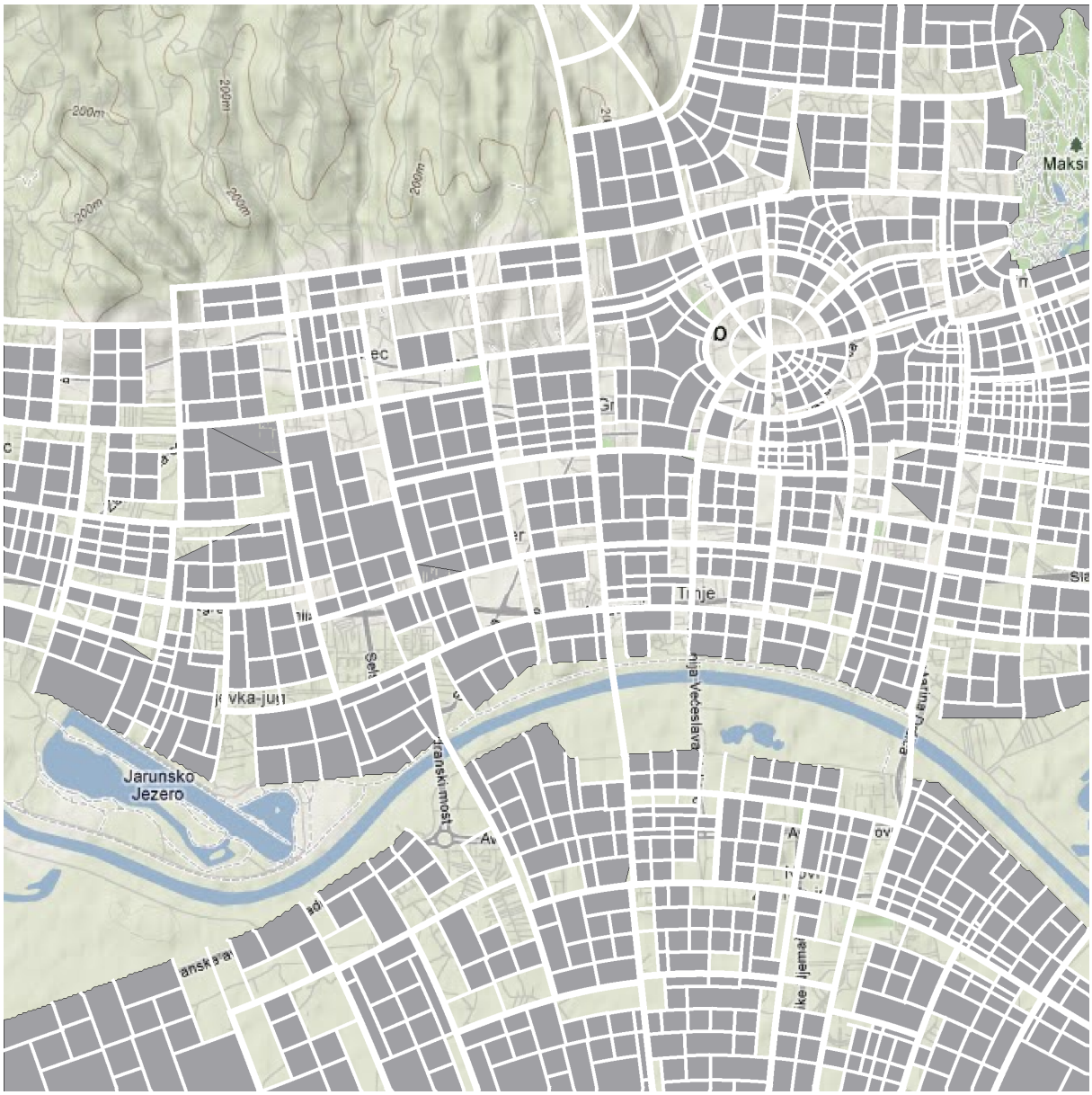


(a) Kombinirana mapa elevacije terena i prirodnih prepreka

(b) Generirana karta ulica

(c) Generirani model grada

Slika 24: Primjer fiktivnog grada na brdovitom terenu



Slika 25: Primjer rekonstrukcije postojećeg grada

7 Zaključak

Tehnike i postupci koje sam istražio u sklopu pisanja ovoga rada i implementirao u programskom dijelu zadatka pokazali su se kao odlična osnova za daljnji razvoj. Pokazuje se mogućnost generiranja velikog broja različitih tipova gradova sa velikim spektrom uzoraka: od planski uređenih gradova koji se baziraju na rešetkastoj strukturi, planskih gradova koji se zasnivaju na skupu koncentričnih prstenova koji se protežu oko gradske središnjice pa sve do kaotičnih gradova nastali pretežno "organskim" rastom.

Kao najzanimljiviji aspekt postupka istaknuo bi činjenicu da je jedan jednostavan i matematički koncizan koncept – koncept simetričnog tenzorskog polja drugog reda – iskorišten za stvaranje kompleksnih i detaljima bogatih rezultata. Programska implementacija opisanih postupaka i tehnika vrlo je jednostavna i čista, bez kompleksne logike koja bi pokrivala rubne ili specijalne slučajeve.

U godinama koje dolaze očekujem znatan porast primjene proceduralnog generiranja sadržaja u industriji specijalnih efekata i računalnih igara. Još jednom bih naglasio da budućnost tehnika za proceduralno modeliranje vidim najviše kao aktivnu pomoć pri modeliranju kompleksnih scena, stavjajući u prvi plan interakciju korisnika (dizajnera) sa proceduralnim algoritmom.

Zahvala

Htio bih zahvaliti svojoj dragoj supruzi što mi nije dopustila da odustanem od studija: *hvala, lv.*

Literatura

[1] *Texturing & Modelling: A Procedural Approach*,
Third Edition,
David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, Steven Worley

[2] *Interactive Procedural Steet Modeling*,
Peter Wonka, Pascal Müller, Eugene Zhang, Guoning Chen, Gregory Esch

[3] *Procedural Modeling of Cities*,
Proceedings of ACM SIGGRAPH 2001,
Yoav I. H. Parish, Pascal Müller

[4] *Procedural Modeling of Buildings*,
Proceedings of ACM SIGGRAPH 2006 / ACM Transactions on Graphics,
Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, Luc Van Gool

[5] *Instant Architecture*,
Peter Wonka, Michael Wimmer, Francçois Sillion, William Ribarsky

[6] *Image-based Procedural Modeling of Facades*,
Proceedings of ACM SIGGRAPH 2007 / ACM Transactions on Graphics,
Pascal Müller, Gang Zeng, Peter Wonka, Luc Van Gool

[7] *The Algorithmic Beauty of Plants*,
Przemyslaw Prusinkiewicz, Aristid Lindenmayer

[8] *Visual Models of Plants Interacting with Their Environment*,
Radomir Mech & Przemyslaw Prusinkiewicz

[9] *Computer Vision*,
Linda Shapiro, George Stockman

[10] *Visualization and Processing of Tensor Fields*,
Joachim Weickert, Hans Hagen (Editors)

[11] *Interactive Tensor Field Design and Visualization on Surfaces*,
IEEE Transactions on Visualization and Computer Graphics,
Eugene Zhang, James Hays, Greg Turk

[12] *Image Based Flow Visualization*,
Jarke J. van Wijk

[13] *Creating Evenly-Spaced Streamlines of Arbitrary Density*,
Bruno Jobard & Wilfrid Lefer

[14] *Polygon Detection from a Set of Lines*,
Proceedings of 12° Encontro Português de Computação Gráfica, Porto, Portugal, 10/2003,
Alfredo Ferreira, Manuel J. Fonseca, Joaquim A. Jorge

[15] *Minimum cycle basis: algorithms and applications (2006)*,
Thesis for obtaining the degree of a Doctor of the Engineering Sciences (Dr.-Ing.) of the natural-technical faculties of Saarland University,
Dimitrios Michail, Dekan Prof and Dr.-ing Thorsten Herfet, Dr. René Beier, Στην Κανέλα

[16] *An Efficient Cycle Vector Space Algorithm for Listing all Cycles of a Planar Graph*,
SIAM Journal on Computing,
Maciej M. Syslo

[17] *A polynomial-time algorithm to find the shortest cycle basis of a graph*,
SIAM Journal on Computing,
Horton, J. D.

[18] *Graph Theory with Applications*,
J.A. Bondy & U.S.R. Murty,
`http://www.ecp6.jussieu.fr/pageperso/bondy/books/gtwa/gtwa.html`

[19] *Wikipedia - Procedural modeling*
17. studenog 2009.,
`http://en.wikipedia.org/wiki/Procedural_modeling`

[20] *Wikipedia - Tenzor*
30. ožujka 2009.,
`http://hr.wikipedia.org/wiki/Tenzor`

[21] *Wikipedia - Region growing*
5. veljače 2010.,
`http://en.wikipedia.org/wiki/Region_growing`

[22] *Wikipedia - Floyd-Warshall algorithm*
23. siječnja 2010.,
`http://en.wikipedia.org/wiki/Floyd-Warshall_algorithm`

[23] *Wikipedia - Cycle space*
22. listopad 2009.,
`http://en.wikipedia.org/wiki/Cycle_space`

Sažetak
(Postupci proceduralnog modeliranja gradova tenzorskim poljima)

Modeli urbanih okruženja od velike su važnosti u mnogim primjenama, od računalnih igara, simulatora letenja, industrije specijalnih efekata, pa sve do projekata urbanog planiranja. Izrada detaljnih modela gradova ili gradskih četvrti može postati iznimno dugotrajan i skup proces. Korištenje proceduralnih tehnika u postupku modeliranja može drastično smanjiti trošak i povećati detaljnost nastalog modela. Ovaj rad prikazati će nekoliko različitih tehnika koje, povezane u jedan zajednički okvir, omogućuju stvaranje detaljnih scena urbanog okruženja.

Ključne riječi: tenzori, tenzorska polja, proceduralno modeliranje, prometne mreže, urbana okruženja

Abstract
(Procedural Modeling of Cities Using Tensor Fields)

Models of urban environments are of great importance in many different applications – from computer games, flight simulators, special effects industry, to the projects of urban planning. Creating detailed models of cities or city districts can be a very time consuming and expensive process. Usage of procedural techniques in the modeling process can both drastically decrease the expense and increase the detail of the created model. This paper presents a number of different techniques that, when combined in a single framework, enable creation of highly detailed scenes of urban enironments.

Keywords: tensors, tensor fields, procedural modeling, street networks, urban environments