SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 62

Reljefne tehnike teksturiranja u prikazu terena

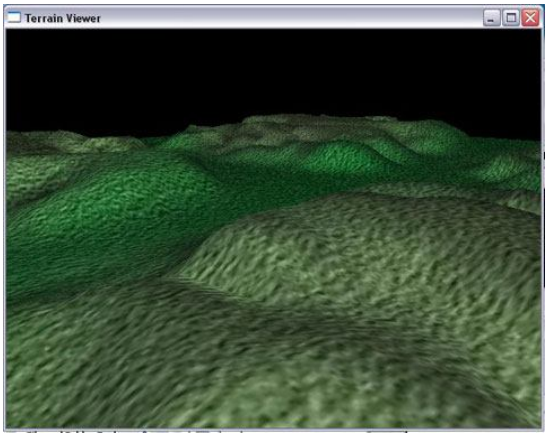
Danijel Janković

Zagreb, lipanj 2010.

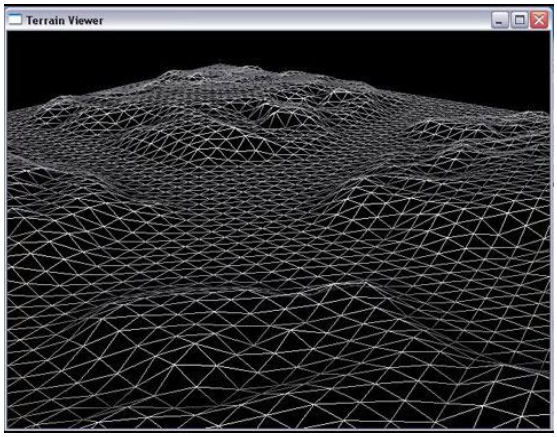
Poglavlja

1. Uvod

Postoji nekoliko načina vizualizacije terena, prvi i poznat stoljećima je „pogled s vrha“ (*engl. Top-down rendering*) kao primjerice na zemljopisnim kartama, drugi koji je omogućen s dobom računalne grafike je perspektivna izrada prikaza (engl. Rendering), tj. pogled na teren iz određenog kuta. Od ranih faza primjene grafike za vojne svrhe, teren je bio važan faktor. Virtualni tereni kreirani računalom su se koristili za uvježbavanje pilota za misije, te kao pomoć zapovjednicima prilikom vizualizacija bojnog polja. Tipična aplikacija za prikaz terena se temelji na: bazi podataka o terenu, dijela koji se obavlja na centralnom (CPU) i dijela koji se obavlja na grafičkom (GPU) procesoru. Dio koji koristi CPU je za identificiranje i dohvaćanje podataka iz baze podataka. Nakon toga primjenjuje potrebne transformacije (pomaci na karti, promjene perspektive…) te gradi mrežu točaka (engl. mesh) koju šalje grafičkom procesoru koji obavlja geometrijske transformacije i kreaciju poligona, nakon čega se iscrta slika terena na ekranu.



Slika 1. Jednostavni 3D teren



Slika 2. Isti teren bez dodane teksture (engl. wireframe)

U modernim grafičkim aplikacijama potrebno je prikazivati foto-realistične terene velikih dimenzija (pogotovo za simulatore leta) s velikom količinom detalja. Problem je taj što procesna moć računala nije neograničena te količina detalja koje računalo može obraditi nije nužno uvijek dovoljna za kvalitetnu iluziju 3D prostora. Konkretno, tereni u simulacijama leta se redovito protežu desetcima ako ne i tisućama kilometara, sa satelitskim slikama kao teksturama radi postizanja efekata foto-realističnosti. Ovaj pristup je funkcionalan i dobar u slučaju ako se simulirani let odvija na velikoj visini s koje detalji na tlu nisu jasno raspoznatljivi. Približavanjem tlu postaje očito da su zgrade na tlu samo teksture „nalijepljene“ na teren, bez ikakve visine. Prvo rješenje ovog problema je dodavanje 3D objekata na teren. Za veće objekte kao zgrade i kuće se dodaju trodimenzionalni poligoni s teksturama a za manje objekte kao aute i manje građevine se koriste teksture.

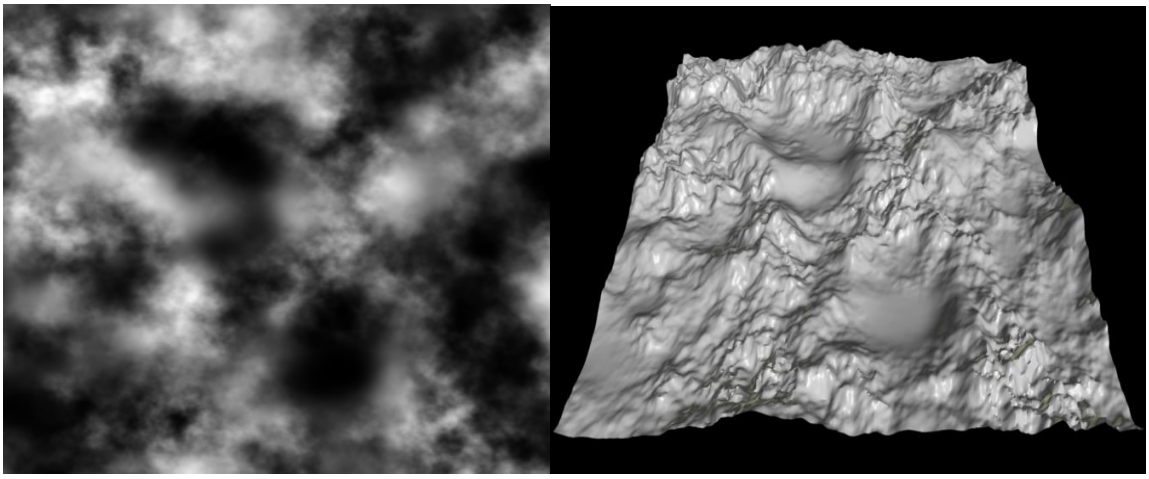
Slika 3. 3D objekti dodani na satelitsku sliku terena

No, problematika tog pristupa je u dodatnoj količini podataka u obliku 3D objekata koje sustav mora obraditi. U slučaju ruralnih terena s rijetkim građevinama to nije problem, ali kod velikih gradova količina podataka dolazi do izražaja. U ovom radu predstavit ćemo tehnike koje imaju potencijal smanjiti zahtjevnost generiranja trodimenzionalnih objekata na srednjim visinama, preslikavanje uz efekt paralakse i reljefno preslikavanje.

2. Prikaz terena

2.1 Visinske Mape (engl. Heightmap)

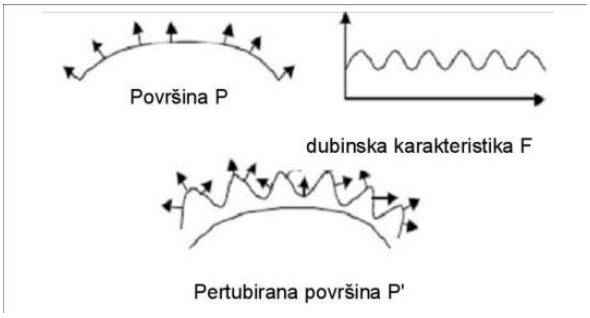
Visinska mapa (drugi naziv visinsko polje) je specifična vrsta slike koja se koristi u 3D grafici, njezina namjena je pohranjivanje podataka o visini točaka u trodimenzionalnom prostoru. Neki od primjera upotrebe visinskih mapa su preslikavanje izbočina (engl. *Bump mapping*), njegovi nasljednici preslikavanje uz efekt paralakse i reljefno preslikavanje, te preslikavanje pomaka (engl. *Displacement mapping*). Visinska mapa sadrži jedan kanal boje koji predstavlja udaljenost pojedine točke od referentne razine, tj. visinu točke. Crna boja najčešće predstavlja najnižu visinu dok bijela boja predstavlja najvišu. Visinske mape se mogu spremati u postojećim slikovnim formatima *(.bmp, .jpeg*) ili specijaliziranim formatima (*Daylon Leveller, GenesisIV, Terragen*).



Slike 4., 5. Visinska mapa (lijevo) i teren kreiran iz iste visinske mape (desno)

2.2 Preslikavanje izbočina (engl. *Bump mapping*)

Tehnika u računalnoj grafici koja se koristi za aproksimaciju hrapavih površina. Ova tehnika se temelji na obavljanju modulacija na normalama površine dobivajući modulirane vektore koje primjenjujemo u proračunu svijetla. Preslikavanje izbočina radi na način da mijenja razinu svjetlosti slikovnog elementa na površini objekta ovisno o visinskoj mapi specificiranoj za svaku površinu. Prilikom iscrtavanja scene svijetlost i boja slikovnih elemenata su određeni interakcijom trodimenzionalnog objekta sa svjetlima u sceni.



Slika 6. Ilustracija modulacija normala kod preslikavanja izbočina

Nakon preslikavanja teksture sljedeće se kalkulacije izvode za svaki slikovni element na površini objekta:

1. Traženje pozicije na dubinskoj karakteristici F koja se podudara s lokacijom na površini
2. Izračun površinske normale visinske mape

$$N(u,v) = \frac{\partial P(u,v)}{\partial u} \times \frac{\partial P(u,v)}{\partial v}$$ Normala u točci P

$$P'(u,v) = P(u,v) + F(u,v)\frac{N(u,v)}{|N(u,v)|}$$ Modulirani vektor pozicije u nad površinom P'

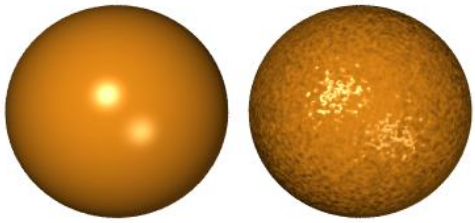
$$N'(u,v) = \frac{\partial P'(u,v)}{\partial u} \times \frac{\partial P'(u,v)}{\partial v}$$ Površinska normala nad površinom P'

3. Dodavanje površinske normale iz 2. koraka geometrijskoj površinskoj normali tako da normala pokazuje u novom smjeru.

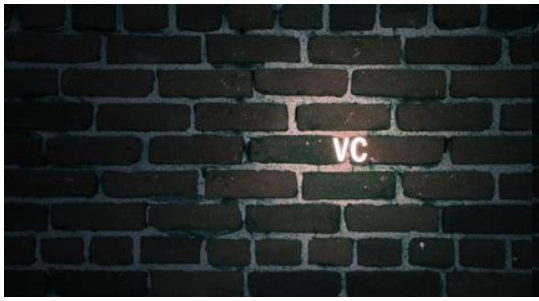
$$N' = N + \frac{\frac{\partial F}{\partial u}\left(N \times \frac{\partial P}{\partial v}\right) - \frac{\partial F}{\partial v}\left(N \times \frac{\partial P}{\partial u}\right)}{|N|}$$

4. Izračunaj interakciju nove „hrapave“ površine sa svijetlima u sceni koristeći neki model osvjetljenja (primjerice Phong Shading). Umjesto pravih normala koriste se modulirane normale.

$$I = k_a\, i_a + \sum_{svijetla} (k_d\,(L \cdot N) i_d) + k_s (R \cdot V)^{\alpha} i_s$$



Slika 6. objekt bez preslikavanja izbočina (lijevo) i sa preslikavanjem izbočina (desno)



Slika 7. Zid bez preslikavanja izbočina

Slika 8. Zid sa preslikavanjem izbočina

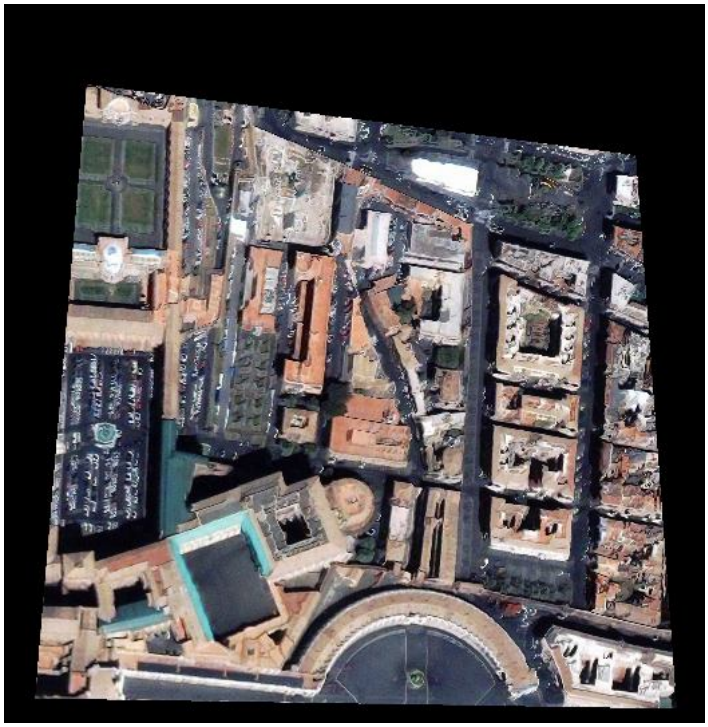
Tehnika preslikavanja izbočina je jednostavna za implementaciju i pruža dobar izgled uz mali pad performansi jer prenosi normale preko tekstura programima za sjenčanje (engl. shader). Sve naprednije tehnike koje su opisane u daljnjem radu su u osnovi stupnjevi poboljšanja preslikavanja izbočina.

2.3 Preslikavanje uz efekt paralakse (*engl. Parallax mapping*)

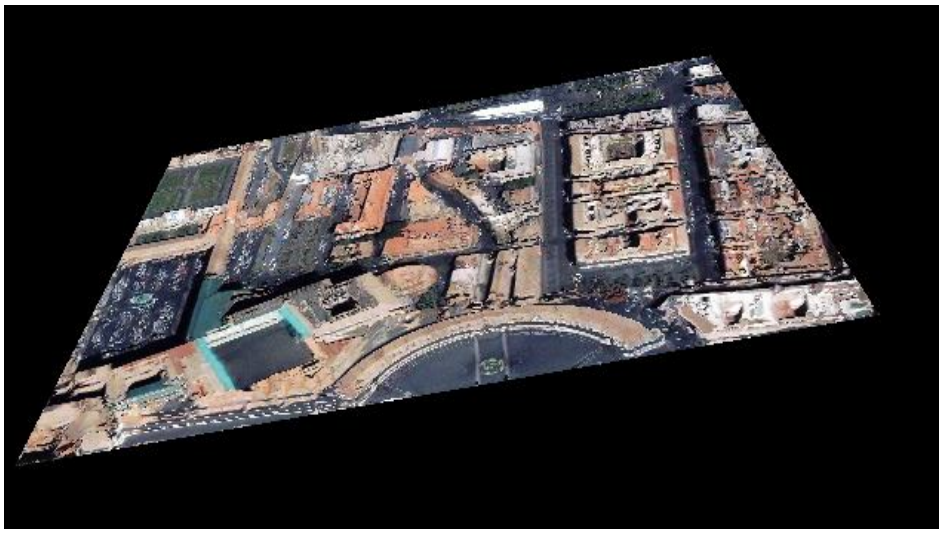
Preslikavanje uz efekt paralakse također znano pod nazivom „preslikavanje uz virtualni pomak“, je unaprijeđena verzija tehnike preslikavanja izbočina (*engl. Bump mapping*) koja dodaje efekt paralakse. Paralaksa je prividni odmak ili razlika u orijentaciji dvaju vektora koji gledaju u isti objekt iz različitih gledišta, mjeri se kutom koji zatvaraju ta dva pogleda. Objekti koji su bliže gledištu imaju veću paralaksu od objekata udaljenijih od gledišta. Krajnji rezultat preslikavanja uz efekt paralakse je izraženija dubinska detaljnost slike. Efekt veće dubinske detaljnosti dobiven je tako da se rade odmaci koordinata teksture nad slikovnim elementom koji se iscrtava, odmaci su dobiveni kao funkcija pogleda i vrijednosti dubinske karakteristike. Dubinsku karakteristiku predstavljaju vrijednosti čiji iznos odgovara dubini koju bi ta točka imala da je površina trodimenzionalna. Ova tehnika je relativno jednostavna i daje odlične rezultate, no ima popriličan broj mana kao primjerice nepostojanje zaklanjanja (engl. occulsion) te deformacija i isticanja plošnosti kod malih kutova gledanja.

Slika 9. Preslikavanje teksture

Slika 10. Preslikavanje teksture uz efekt paralakse



Slika 11. Preslikavanje uz efekt paralakse s većim zadovoljavajućim kutom gledanja



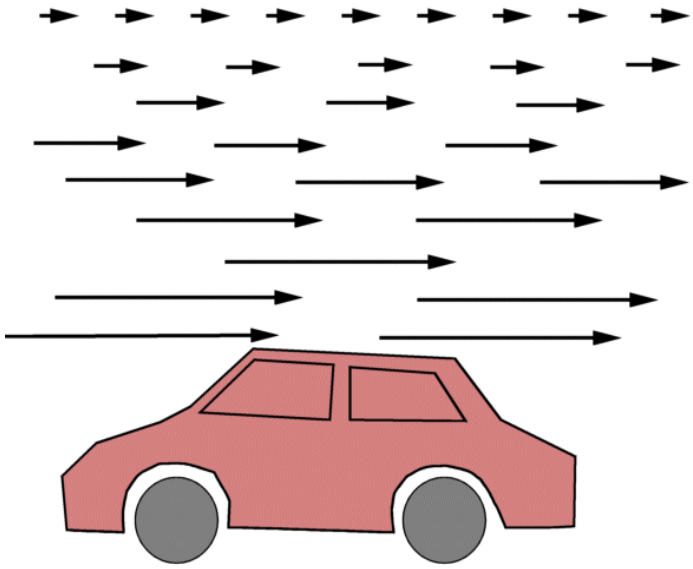
Slika 12. Deformacije i isticanje plošnosti kod manjih kutove gledanja.

2.3.1 Vrste paralakse

Obzirom na definiciju paralakse razlikujemo 2 pogleda na pojam:

1. Paralaksa dobivena gibanjem (*engl. Parallax scrolling*)

   Iluzija dubine koja se dobiva kao rezultat gibanja promatrača u odnosu na objekt, koristi se činjenicom da objekti koji su bliži promatraču se brže gibaju njegovim vidnim poljem. U mnogim ranim grafičkim aplikacijama scena se podijelila na slojeve koji su se gibali različitim brzinama kako bi se dobila iluzija dubine prostora.

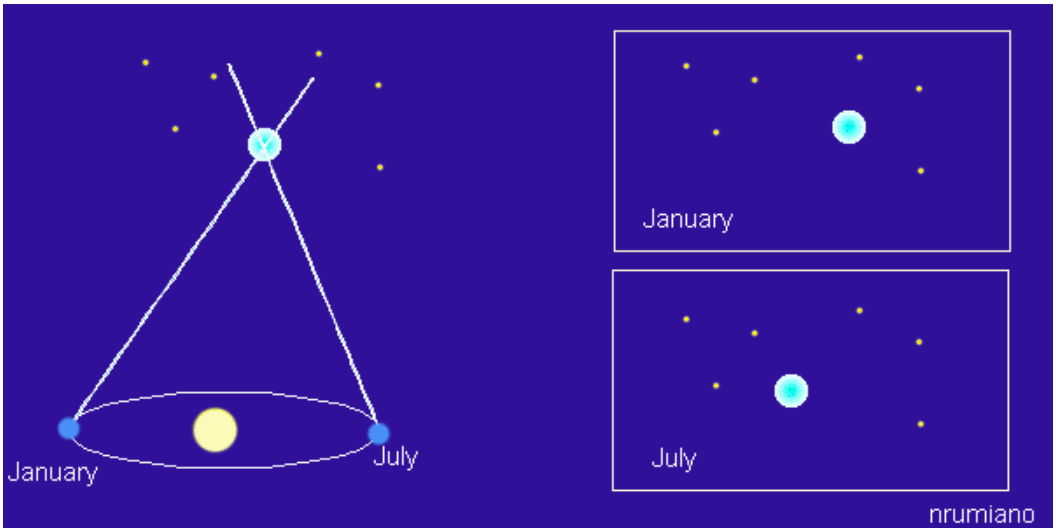


Slika 13. Paralaksa uzrokovana gibanjem, udaljeniji objekti(kraće strelice) se kreću sporije od onih bližih promatraču

2. Paralaksa dobivena zbog promjene kuta između vektora pogleda i površine

   Kada se neki predmet promatra iz 2 različita kuta, slika predmeta nije ista, razlog toga je što iz različitih kutova sjecišta našeg vektora gledanja i

promatranog objekta nisu ista. Određena područja su više sužena ili proširena ovisno o kutu promatranja. Primjer ovog efekta paralakse je gledanje u zvijezde od strane astronoma na različitim dijelovima Zemlje.

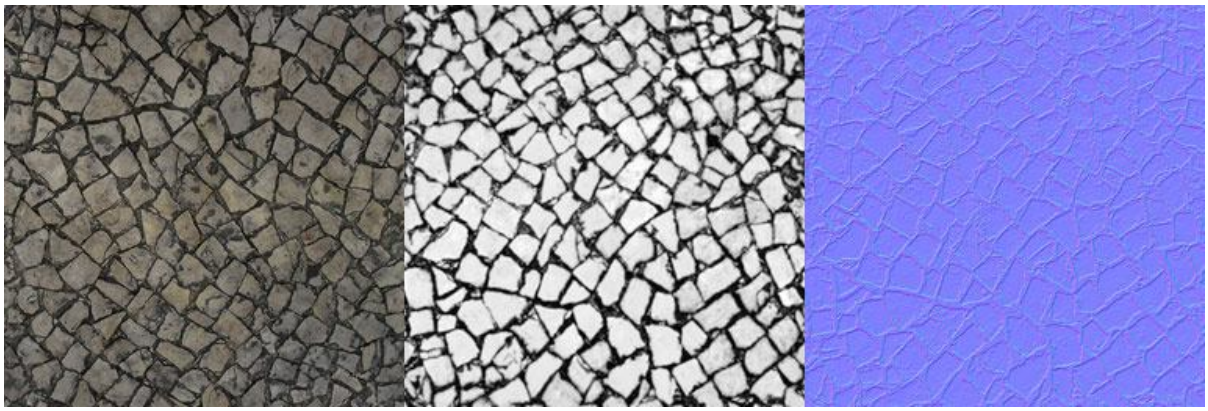


Slika 14. Primjer efekta paralakse u astronomiji (promatrana zvijezda se nalazi na drukčijoj poziciji ovisno o tome u kojem mjesecu ju promatramo)

Preslikavanje uz efekt paralakse koristi 2 ulazna parametra:

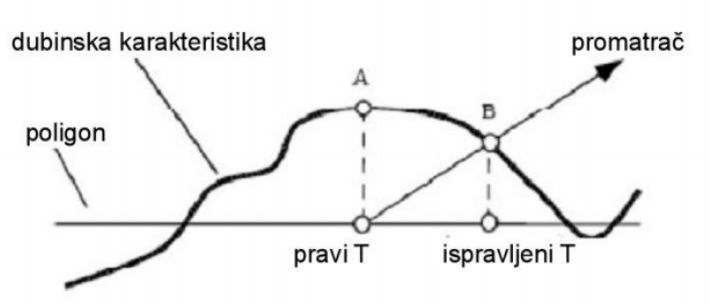
1. Mapu normala (Koju koristi i preslikavanje izbočina)
2. Dubinsku mapu

Te dvije vrijednosti se spremaju u jednu teksturu na taj način da RGB komponenta sadrži mapu normala, dok alfa (*engl. alpha*) komponenta sadrži dubinsku karakteristiku.



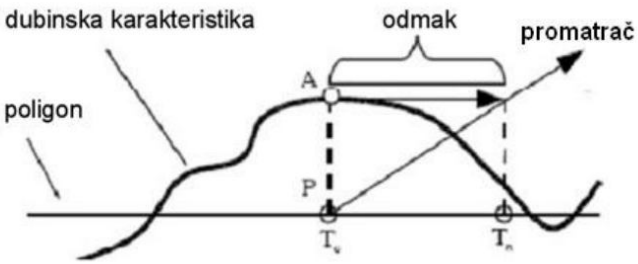
Slika 15. (s lijevo na desno) Tekstura, dubinska karakteristika, mapa normala

Sam algoritam preslikavanja uz efekt paralakse se temelji na računanju odmaka koordinata na danom slikovnom elementu (engl. pixel), proračun se izvodi u sustavu tangente kao i kod preslikavanja izbočina. Z komponenta predstavlja visinu, tj. okomitu udaljenost od površine, dok x i y komponente definiraju pomake unutar površine.



Slika 16. Greška u prikazu odgovarajućeg slikovnog elementa

Obzirom da se dubinska karakteristika razlikuje od površine poligona promatrač vidi slikovni element A (pravi T) umjesto elementa B (ispravljeni T). Preslikavanje uz efekt paralakse aproksimira potrebni odmak da bi promatrač vidio ispravni slikovni element .



Slika 17. Aproksimacija odmaka preslikavanjem uz efekt paralakse

Navedeni odmak (Tn) se računa na sljedeći način : $$T_n = T_v + h' \frac{V_{\{x,y\}}}{V_{\{z\}}}$$

Tn - nove koordinate teksture

Tv - originalne koordinate teksture (one koje bi promatrač vidio da ne postoji efekt paralakse)

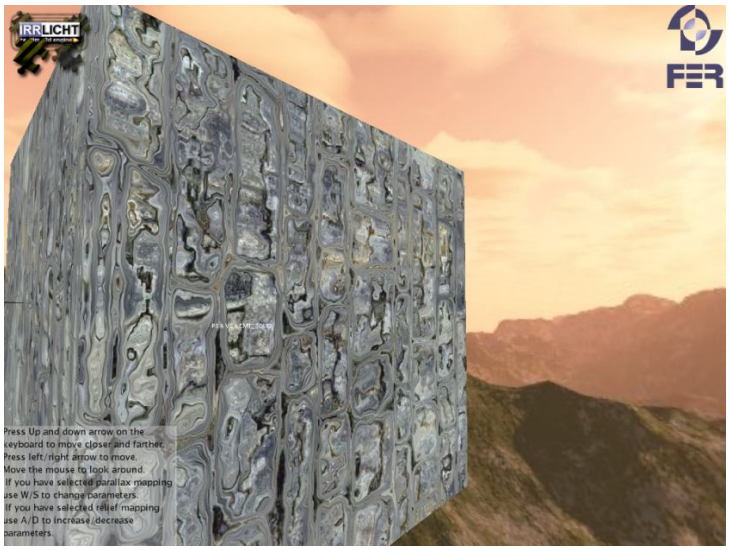
V – vektor pogleda od promatrača prema slikovnom elementu (definiran u tangentnom sustavu)

h' – skalirana verzija vrijednosti dubinske karakteristike

Slika 18. objašnjenje jednadžbe paralakse

Dodavanjem umnoška h' i rezultata dijeljenja V(x,y)/V(z) dobivamo željeni odmak na teksturi, koji dodajemo originalnim koordinatama teksture. Dubinska vrijednost h' se skalira jer je u granicama [0,1] a korištenje te vrijednosti bi rezultiralo preklapanjem i međusobnim ispravljanjem određenih dijelova teksture što bi izgledalo kao slijevanje slikovnih elemenata.

Ovaj algoritam je jednostavan i brz, no ima nekoliko velikih nedostataka, prvi je taj da nema samo-zaklanjanja, tj. algoritam ne vodi računa o tome ako jedan dio teksture zaklanja neki drugi. Dolazi do preklapanja i isprekidanosti teksture zbog aproksimacije odmaka. Također ako promatrač gleda teksturu iz nekog malog kuta, slijevanje slikovnih elemenata postaje primjetno zbog strmog nagiba u dubinskoj karakteristici. Ove greške su posljedica činjenice da algoritam pretpostavlja da je cijela površina poligona ravna, tj. da ima konstantnu dubinsku vrijednost. Ovo je posebno problematično kod neravnih površina.



Slika 19. Slijevanje teksture kod preslikavanja uz efekt paralakse

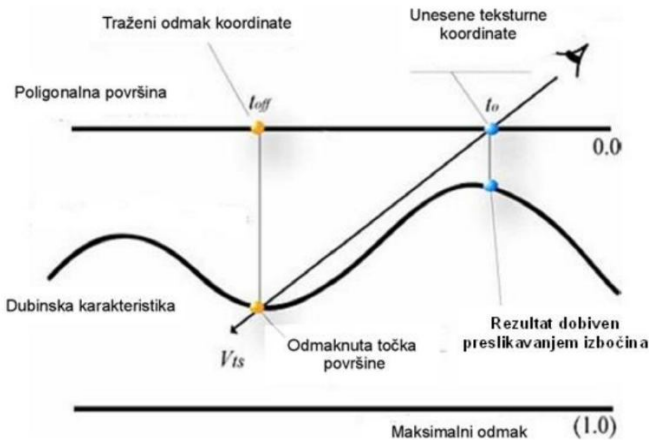
2.3.2 Poboljšanje algoritma

Preslikavanje uz efekt paralakse s ograničenjem odmaka (*engl. parallax mapping with offset limiting*). Odstranjivanjem 1/z dijela jednadžbe paralakse postiže se ograničavanje maksimalnog pomaka na h' u oba smjera teksture čime se postiže smanjivanje grešaka dobivenih preklapanjem.

Modificirana jednadžba

X i Y koordinate vektora pogleda predstavljaju mjeru skaliranja te zbog toga vektor pogleda mora biti normaliziran.

$$T_n = T_v + h' * V_{\{x,y\}}$$



Slika 20. Presjek vektora pogleda i dubinske karakteristike, u slučaju nepostojanja efekta paralakse promatrač bi vidio plosnatu strukturu, s efektom paralakse promatrač vidi točku koju bi gledao u slučaju da je površina trodimenzionalna.

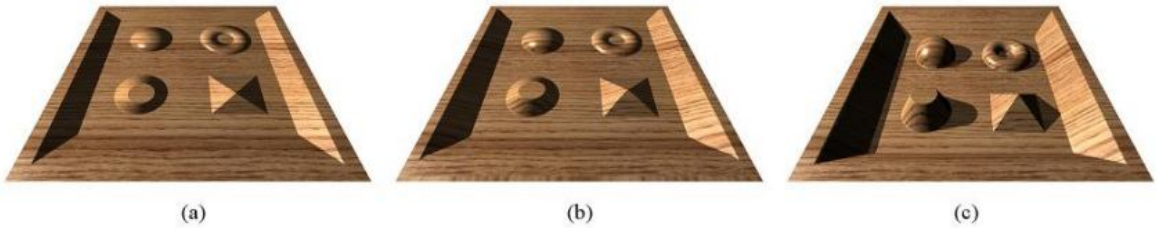
2.3.3 Tehnike preslikavanja uz efekt paralakse praćenjem zrake pogleda

Tehnike preslikavanja uz efekt paralakse praćenjem zrake pogleda je naziv za skupinu algoritama koji koriste tehniku praćenja zrake radi preciznijeg određivanja sjecišta zrake pogleda i dubinske karakteristike. Ove tehnike poboljšavaju efekt paralakse tako da poboljšavaju aproksimativno računanje odmaka. Tehnike efekta paralakse s praćenjem zrake se temelje na iterativnom traženju sjecišta vektora pogleda i dubinske karakteristike.

U ovu skupinu spadaju reljefno preslikavanje (engl. *Relief mapping*), preslikavanje prekrivanja uslijed paralakse (engl. *Parallax occlusion mapping*) i preslikavanje strminom uz efekt paralakse (engl. *Steep parallax mapping*). Ove tehnike su slične te se razlikuju u načinu određivanja sjecišta. U nastavku ćemo obraditi reljefno preslikavanje te njegove prednosti i mane.

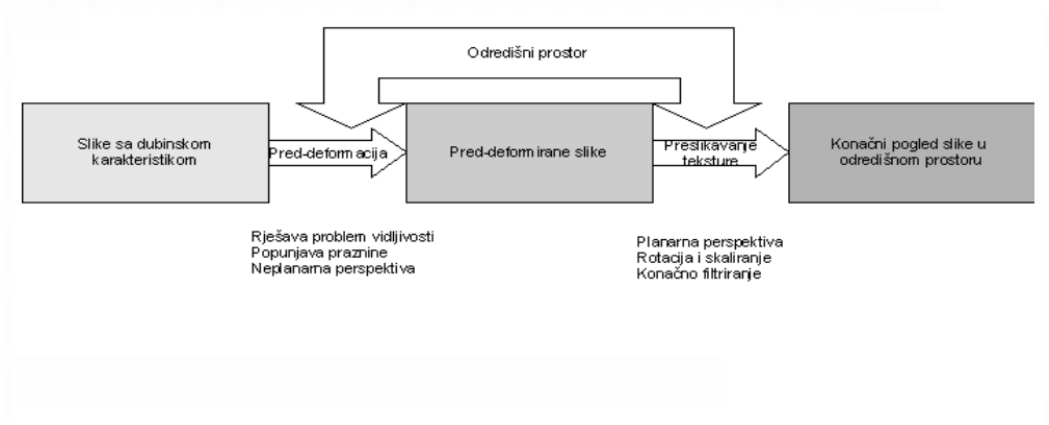
2.4 Reljefno preslikavanje

Ova tehnika daje efekt paralakse uzrokovan pogledom i kretanjem na način da se tekstura deformira (engl. *pre-warp*) prije samog preslikavanja na površinu. Ulaz u algoritam je dubinska karakteristika slike i vektor pogleda, a pred-deformacija je odgovorna za rješavanje problema vidljivosti i popunjavanje praznina. Izlazna slika algoritma se koristi kao ulaz u preslikavanje teksture koje stvara krajnju sliku.



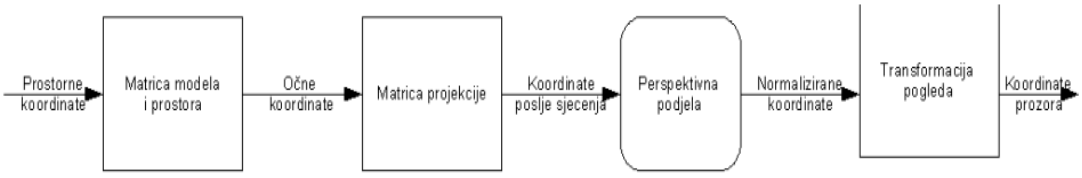
Slika 21. Usporedba tehnika a)preslikavanje izbočina, b)preslikavanje uz efekt paralakse, c)reljefno preslikavanje

Slika 22. Još jedan primjer razlike preslikavanja reksture i reljefnog preslikavanja



Slika 23. Algoritam za reljefno preslikavanje teksture

2.4.1 Trodimenzionalna deformacija teksture



Slika 24. Put podataka kroz grafički protočni sustav

Koraci

1. Objekt je dan u svom koordinatnom prostoru i matricom modela

$$P_{object} = [x_0, y_0, z_0, 1]^T \qquad M_{modelview}$$

2. Potrebno je prebaciti te koordinate u sustav oka ($P_{eye}=[x_{eye}, y_{eye}, z_{eye}, w_{eye}]^T$), to se radi sljedećim izrazom $P_{eye}= M_{modelview}\ P_{object}$, tj. množenjem matrice modela i koordinatnog sustava objekta.
3. Slično kao i u drugom koraku za dobivanje koordinata poslije sječenja (engl. clipping) $P_{clip} = [x_{clip}, y_{clip}, z_{clip}, w_{clip}]^T$ koristimo izraz $P_{clip}= M_{projection}P_{eye}$,tj. množenje matrice projekcije i koordinatnog sustava oka. Te koordinate je potrebno normalizirati, to se radi izrazom $P_{NDC} \overset{\div w}{=} P_{clip}$ (simbol $\overset{\div w}{=}$ označava jednakost poslije operacije dijeljenja homogenim faktorom w s desne strane). Važno je napomenuti da su X, Y, Z komponente od Pndc skalirane u granicama [-1,1].

Uzmimo da su dimenzije prozora u kojem se iscrtava (engl. viewport) W (širina) i H (dužina), te da je lokacija ishodišta u centru gledanja, dubinski opseg je [n, f]. PNDC se projicira u točku unutar prozora gledanja preko transformacijske matrice (Mviewport), ta točka iznosi $P_{win} = [x_{win}, y_{win}, z_{win}, 1]^T$ i dobiva se preko izraza Pwin = Mviewport * Pndc.

$$M_{\text{viewport}} = \begin{bmatrix} W/2 & 0 & 0 & W/2 \\ 0 & H/2 & 0 & H/2 \\ 0 & 0 & (f-n)/2 & (f+n)/2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

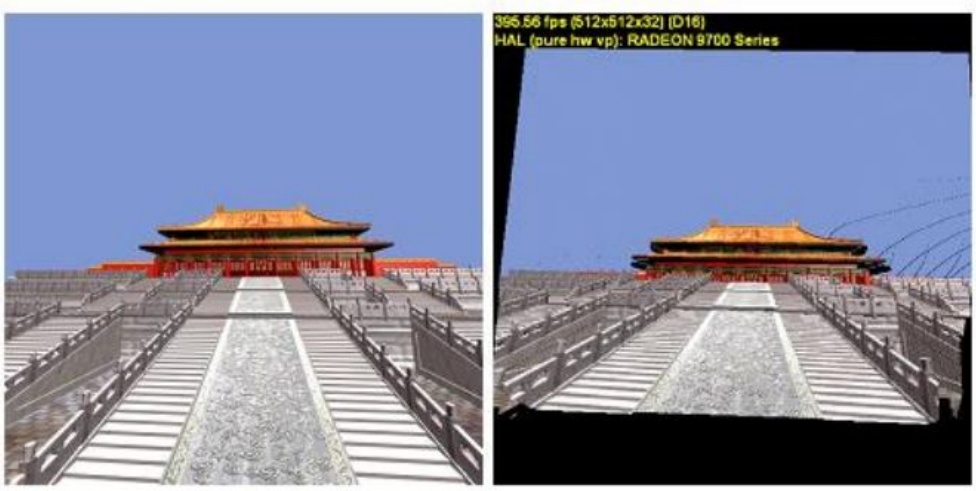
Radi ubrzavanja operacija sve se matrice mogu spojiti u jednu matricu :
Mn = Mviewport * Mprojection * Mmodelview

Izraz za točku unutar prozora gledanja koji generira referentnu sliku sada je $P_{win} \overset{\div w}{=} M_n P_{object}$

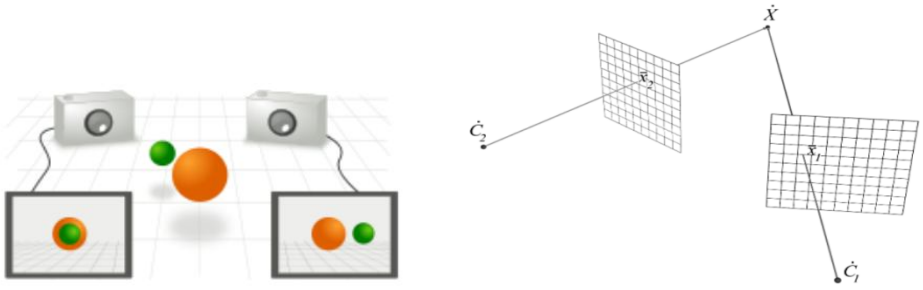
Nakon toga sama transformacija unutar trodimenzionalne deformacije slike se odvija u dva koraka:

1. Vratiti točke unatrag u lokalne koordinate objekta (inverzno preslikavanje)
2. Preslikati dobivene točke u drugo gledište (engl. viewpoint) kroz isti protočni sustav

$$M_{warping} = M_{d\pi} M_{s\pi}^{-1}$$

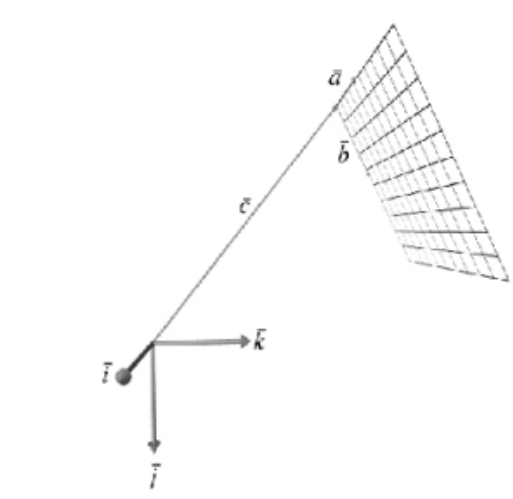


Slika 25: Lijevo referentna slika, desno deformirana slika nakon 3dimenzionalne deformacije



Slika 26. Primjer razlike u pogledu na scenu ovisno o poziciji kamere (promatrača)

2.4.2 Formalni opis trodimenzionalne deformacije teksture



Slika 27. Odnos slikovnog prostora i odredišnog vektora

Planarno projeciranje točke u prostor slike opisan ravninom u 3D prostoru se radi sljedećom funkcijom.

$$\bar{d} = \begin{bmatrix} d_i \\ d_j \\ d_k \end{bmatrix} = \begin{bmatrix} a_i & b_i & c_i \\ a_j & b_j & c_j \\ a_k & b_k & c_k \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{P} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Vektor $\bar{d}$ je vektor koji polazi od izvorišta koordinatnog sustava i prolazi kroz točku (u,v) unutar prostora slike. Vrijednosti unutar matrice P čine vektori a, b (koji razapinju dvodimenzionalni prostor slike) i vektor c (vektor iz izvorišta sustava u početnu točku ravnine slikovnog prostora).

2.4.3 Geometrijski model scene

Geometrijski model scene opisan je kombinacijom dubinske karakteristike i modelom kamere asociranim s izvornom slikom.

x - točka u euklidskom prostoru čija projekcija na slikovnu ravninu is ima koordinate (us ,vs). Projekcija točke x na neku proizvoljnu ravninu id se obavlja na sljedeći način :

$$\overrightarrow{x_t} \triangleq P_t^{-1} P_s \overrightarrow{x_s} + P_t^{-1} \left(C_s \dot{-} \dot{C}_t \right) \delta_s(u_s, v_s)$$

$\triangleq$ označava jednakost čijem rezultatu nedostaje jedan skalarni produkt parametara.

i_t - predstavlja odredišni prostor (*target space*).

i_s - predstavlja referentni prostor (*source space*).

Matrice P_t, P_s – predstavljaju 3x3 matrice kamere asocirane sa prostorom i_t odnosno i_s.

$x_s = [u_s, v_s, 1]^T$ $ix_t = [u_t, v_t, 1]^T$ predstavljaju točke (u, v) u izvorišnoj slici i_s , odnosno odredišnoj i_t.

C_s i C_t su centri projekcije točkastih kamera asocirane sa prostorom i_s odnosno i_t.

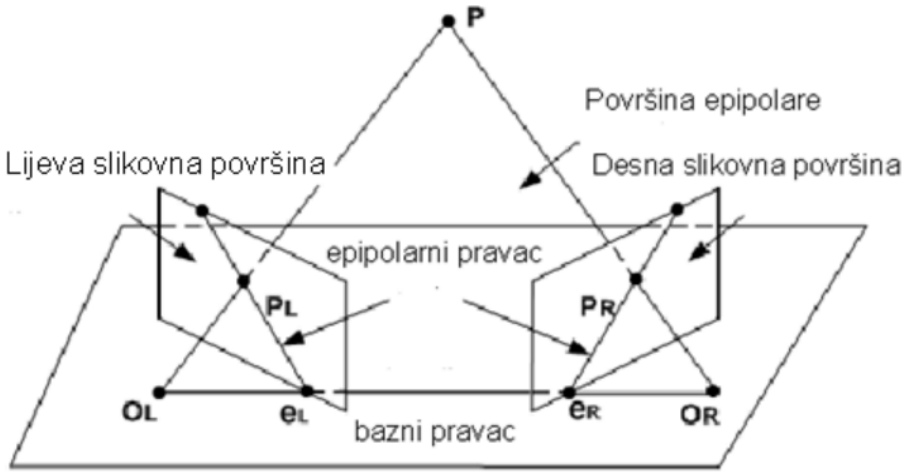
$\delta_s(u_s, v_s)$ je generalno nepodudaranje izvornog slikovnog elementa (u_s, v_s).

Iz gornje relacije se vidi da je odredišnu sliku moguće dobiti primjenom planarno-perspektivne transformacije nad izvornom slikom nakon koje se obavlja proporcionalni pomak točaka po slikovnom pravcu u smjeru epipolarnog pravca $\delta_s(u_s, v_s)$. Epipolarni pravac je spojnica sjecišta linije koja spaja pozicije točkaste kamere sa slikovnom prostornom ravninom i točke x koja daje sjecište odgovarajuće kamere i svoje prostorne slikovne ravnine ik.

Ovakva se faktorizacija naziva površina-plus-paralaksa (engl. plane-plus-parallax). Preslikavanje teksture se može smatrati specijalnim slučajem trodimenzionalne slikovne deformacije gdje sve točke dijele jednako nepodudaranje. Reljefno preslikavanje teksture uvodi faktorizaciju jednadžbe kako bi se poslije procesa deformacije ispravno primijenilo preslikavanje teksture.

Prednosti ovog načina su:

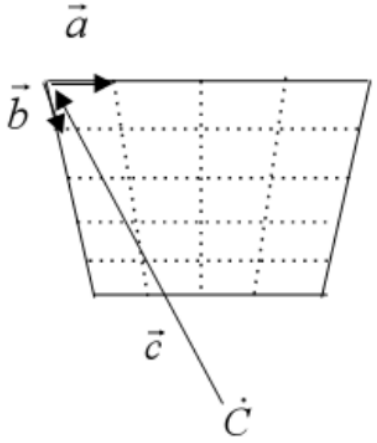
- upotreba grafičkog sklopovlja za operacije preslikavanja teksture, ono preuzima izvođenje transformacija i filtriranja
- prikaz deformacije kao jednodimenzionalne slikovne operacije koja se vrši po redcima i stupcima zahtijevajući samo 2 susjedna elementa teksture
- prirodna integracija tehnike s grafičkim sučeljem



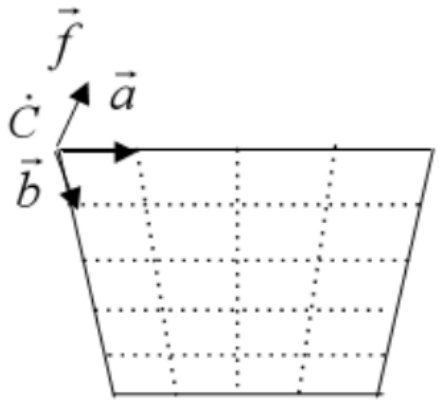
Slika 28. Dobivanje epipolarnog pravca određivanjem sjecišta spojnice dviju točkastih kamera i ravnine slike

2.4.4 Pred-deformacija

Prije dodavanja idealne faktorizacije potrebno je prethodno pronaći „pred-deformaciju“ *p*, kompozicija iste te preslikavanja teksture *m* je ekvivalentna prethodno definiranoj trodimenzionalnoj slikovnoj deformaciji *w*, drugim riječima *w = m * p.* Tokom same deformacije elementi se pomiču okomito i vodoravno u prostoru teksture, vrijednosti pomaka ovisi o visinskoj karakteristici i karakteristikama kamere. U tehnici reljefnog preslikavanja tekstura umjesto perspektivne točkaste kamere koristi se model paralelne projekcijske kamere.



Slika 28. Perspektivna točkasta kamera



Slika 29. Paralelna projekcijska kamera

Uzimajući u obzir paralelno projekcijski model kamere točka $\dot{x}$ se dobiva iz sljedećeg izraza:

$$\dot{x} = \dot{C}_s + \begin{bmatrix} a_{sı} & b_{sı} & f_{sı} \\ a_{sJ} & b_{sJ} & f_{sJ} \\ a_{sk} & b_{sk} & f_{sk} \end{bmatrix} \begin{bmatrix} u_s \\ v_s \\ displ\,(u_s, \quad v_s) \end{bmatrix} = \dot{C}_s + P_s' \vec{x}_s'$$

- (ui ,vi) = (us +Δu, vs +Δv) predstavljaju među koordinate dobivene pomakom originalnih koordinata pomak (Δu, Δv)

- Vektori a i b predstavljaju bazu za slikovni prostori i.

- Vektor f u tom paralelnom projekcijskom modelu kamere predstavlja vektor okomit na slikovni prostor i

- $\dot{C}_s$ predstavlja ishodište slikovne prostorne ravnine

- displt(us,vs) je ortogonalni pomak definiran dubinskom karakteristikom točke (us,vs)

Projekcija koordinata iz izvorišnog sustava is u odredišnu perspektivnu projekcijsku kameru obavlja se sljedećim izrazom :

$$u_t = \frac{Au_s + Bv_s + D + C'displ(u_s, v_s)}{Iu_s + Jv_s + L + K'displ(u_s, v_s)}$$

$$v_t = \frac{Eu_s + Fv_s + H + G'displ(u_s, v_s)}{Iu_s + Jv_s + L + K'displ(u_s, v_s)}$$

$$A = \overrightarrow{a_s} \cdot (\overrightarrow{b_t} \times \overrightarrow{c_t}), B = \overrightarrow{b_s} \cdot (\overrightarrow{b_t} \times \overrightarrow{c_t}), C = \overrightarrow{f_s} \cdot (\overrightarrow{b_t} \times \overrightarrow{c_t}),$$
$$D = (\dot{C}_s - \dot{C}_t) \cdot (\overrightarrow{b_t} \times \overrightarrow{c_t}), E = \overrightarrow{a_s} \cdot (\overrightarrow{c_t} \times \overrightarrow{a_t}), F = \overrightarrow{b_s} \cdot (\overrightarrow{c_t} \times \overrightarrow{a_t}),$$
$$G' = \overrightarrow{f_s} \cdot (\overrightarrow{c_t} \times \overrightarrow{a_t}), H = (\dot{C}_s - \dot{C}_t) \cdot (\overrightarrow{c_t} \times \overrightarrow{a_t}), I = \overrightarrow{a_s} \cdot (\overrightarrow{a_t} \times \overrightarrow{b_t}),$$
$$J = \overrightarrow{b_s} \cdot (\overrightarrow{a_t} \times \overrightarrow{b_t}), K' = \overrightarrow{f_s} \cdot (\overrightarrow{a_t} \times \overrightarrow{b_t}), L = (\dot{C}_s - \dot{C}_t) \cdot (\overrightarrow{a_t} \times \overrightarrow{b_t})$$

Odgovarajući izrazi za preslikavanje teksture dobivaju se iz prethodnih izraza tako se *displt(us,vs)* izjednači s nula za sve slikovne elemente (engl. *pixel*) originalne slike.

$$\frac{Au_i + Bv_i + D}{Iu_i + Jv_i + L} = \frac{Au_s + Bv_s + D + C'displ(u_s, v_s)}{Iu_s + Jv_s + L + K'displ(u_s, v_s)}$$
$$\frac{Eu_i + Fv_i + H}{Iu_i + Jv_i + L} = \frac{Eu_s + Fv_s + H + G'displ(u_s, v_s)}{Iu_s + Jv_s + L + K'displ(u_s, v_s)}$$

Rješavanjem navedenih izraza dolazi se do vrijednosti za koordinate dobivene pomakom (ui, vi):

$$u_i = \frac{u_s + k_1\, displ(u_s, v_s)}{1 + k_3\, displ(u_s, v_s)}$$

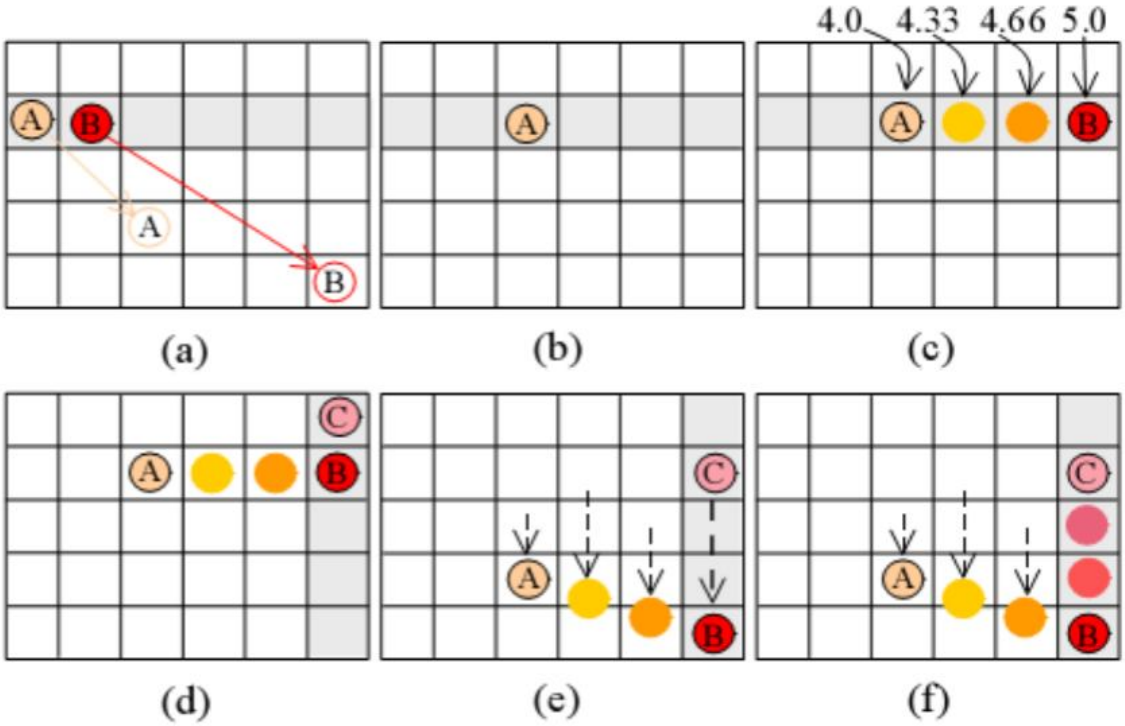
$$v_i = \frac{v_s + k_2\, displ(u_s, v_s)}{1 + k_3\, displ(u_s, v_s)}$$

k1,k2,k3- konstante koje predstavljaju karakteristike odredišne i izvorne kamere, u kombinaciji s *displt(us,vs)*

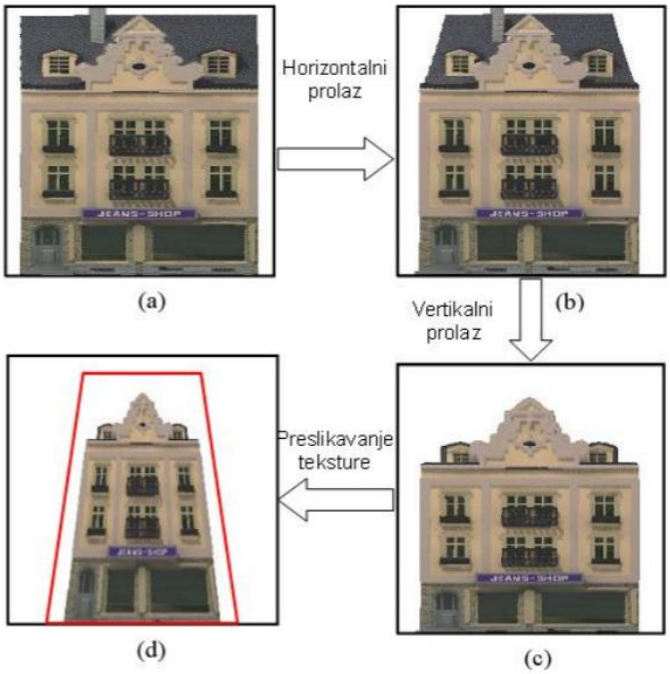
čine pomak (Δu, Δv) u izvornim elementima teksture.

2.4.5 Rekonstrukcija pred-deformirane teksture

Računanje među-točaka je prva, jednostavnija faza pred-deformacije, druga faza je računski puno zahtjevnija rekonstrukcija podataka u tablicu slikovnih elemenata međuslike. Ova faza rješava problem neželjenih učinaka tijekom diskretizacije (engl. *anti-aliasing*), zbog specifične strukture pred-deformacijske relacije, rekonstrukcija je implementirana kao dvofazni proces. Sastoji se od 2 prolaza, horizontalnog i vertikalnog.



Slika 30. Deformacije jednog elementa teksture a)elementi teksture u izvornoj teksturi i njihovi ekvivalenti nakon pred-deformacije. b)prvi element teksture u trenutačnom retku se prebacuje u svoj krajnji stupac c)drugi element teksture se prebacuje u svoj krajnji stupac i vrši se interpolacija boje tijekom rasterizacije. d)nakon što su svi reci deformirani, element teksture C se prilagođava prema elementu teksture B e)duž svih stupaca elementi teksture se pomiču prema krajnjim recima. f)Boja se interpolira tokom rasterizacije.



Slika 31. Faze reljefnog preslikavanja teksture

Pseudokod algoritma pred-deformacije (C-boja, D-odmak, (U,V) – koordinate teksture):

dohvati U_{in}, V_{in}, C_{in}, D_{in}

U_{next} = *relacija_za izracun_ui*(U_{in}, D_{in})

V_{next} = *relacija_za izracun_vi*(V_{in}, D_{in})

za(U_{out} = *integer* (U_{prev} + 1); U_{out} <= U_{next}; U_{out}++){

linearno interpoliraj C_{out} *izmedu* C_{prev} *i* C_{in}

linearno interpoliraj V_{out} *izmedu* V_{prev} *i* V_{in}

postavi C_{out} *i* V_{out} *kod* U_{out}

U_{prev} = U_{next} ; V_{prev} = V_{next}; C_{prev} = C_{next};

}

2.4.6 Implementacija reljefnog preslikavanja u realnom vremenu

Reljefno preslikavanje tangente se obavlja u vektorskom prostoru tangente, zbog toga se tehnika može primijeniti nad zakrivljenim površinama proizvodeći ispravno samo-zatamnjenje, sjene i osvjetljenje obavljeno nad slikovnim elementom. Sama tehnika se zasniva na algoritmu za određivanje sjecišta dubinske karakteristike i vektora pogleda koji se temelji na inverznom postupku (gledamo iz slikovnog elementa) praćenja zrake.

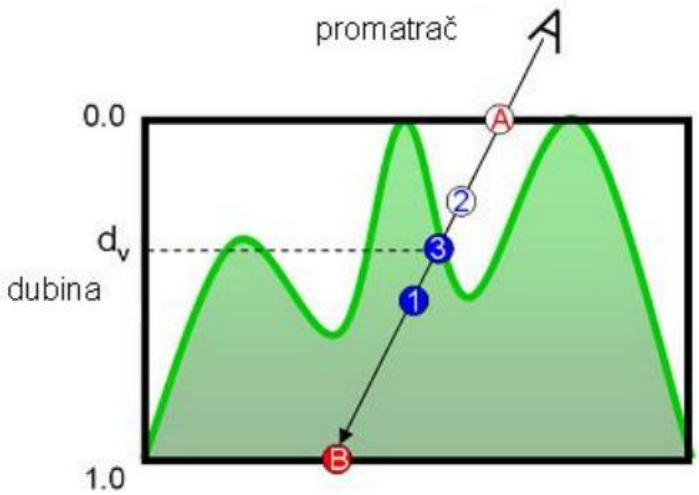
Reljefno preslikavanje teksture na proizvoljnu površinu se radi na sljedeći način:

Za svaki fragment koji će biti iscrtan:

- *izračunaj vektor pogleda (engl. VD-viewing direction), tj. vektor od gledišta do točke na proizvoljnoj poligonalnoj površini.*
- *prebaci vektor pogleda VD u vektorski prostor tangente koji je asociran s trenutačnim fragmentom koji se obrađuje*
- *koristi VD' (transformirani VD) i A, (s,t) koordinatu teksture fragmenta koji se trenutno obrađuje, kako bi izračunao B,(u,v) koordinatu teksture gdje zraka dohvaća dubinsku vrijednost 1.0.*
- *izračunaj sjecište između VD' i dubinske karakteristike koristeći algoritam za binarno pretraživanje počevši od A do B.*
- *obavi sjenčanje fragmenta koristeći atribute (normala, boja, visina) asocirane s koordinatom teksture koja se dobije iz prethodno izračunatog sjecišta.*

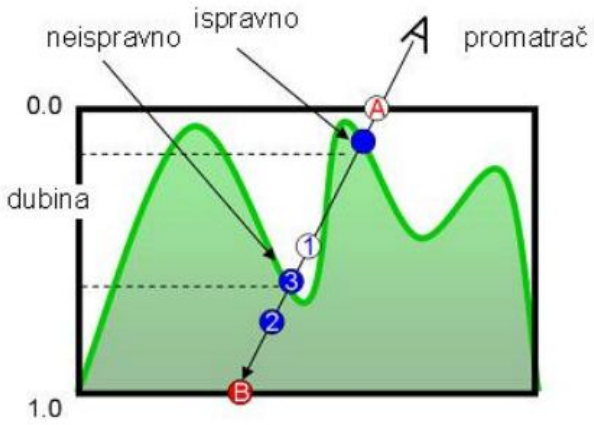
2.4.7 Binarno pretraživanje

Dubinska vrijednost točke A je 0.0 dok je dubinska vrijednost točke B 1.0. U svakom se koraku algoritma računa središnja točka za dani interval, te joj se dodjeli aritmetička vrijednost dubina i koordinata teksture krajnjih točaka. Središnja vrijednost koordinata teksture služi za dobivanje dubinske vrijednosti u teksturi s dubinskim karakteristikama. Ako je spremljena vrijednost manja od izračunate, točka duž zrake je unutar visinske površine kao u slučaju točke 1. Binarno pretraživanje nastavlja svoje izvođenje dok god je jedna krajnja vrijednost unutar, a druga izvan visinske površine. Broj unutar kruga označava redoslijed unutar kojeg su dohvaćene vrijednosti. U praksi se često događa da je oko 8 iteracija dovoljno za dobivanje prihvatljivih rezultata (8 iteracija označava podjelu dubinskog opsega na 256 jednakih dijelova, 2^8 = 256).



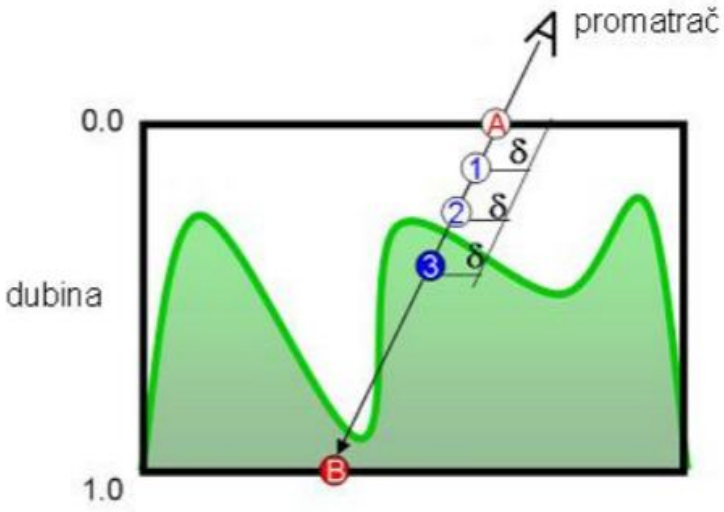
Slika 32. Nalaženje sjecišta vektora pogleda i dubinske karakteristike koristeći binarno pretraživanje. Pretraživanje ide od točke A prema B. Brojevi označavaju iteracije pri kojima se računaju središnje točke.

Problem kod binarnog pretraživanja je taj što postoji mogućnost da središnja točka bude izračunata kao lokacija izvan visinske površine. Kao i na prethodnoj slici, izračunate središnje vrijednosti su redom 1, 2, 3, dolazi do grešaka ako upadna zraka siječe dubinsku karakteristiku u više točaka.



Slika 33. Problem kod binarnog pretraživanja središnja točka od A i B izvan visinske površine

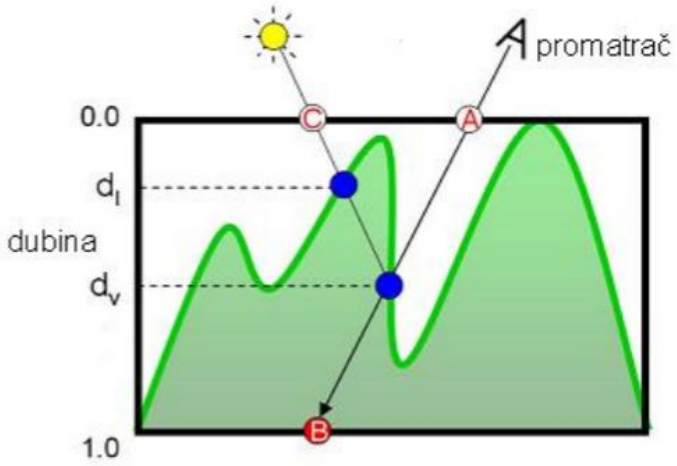
Radi izbjegavanja navedenog problema algoritam počinje sa linearnim pretraživanjem, pretraživanje počinje o točke A, vršeći pomake od δ puta dužine AB tražeći prvu točku presjeka. To se pretraživanje može izvoditi paralelno uz druge operacije pošto ne postoji uvjetni pristup koordinatama teksture. Nakon što je pronađeno prvo sjecište linearnim pretraživanjem, binarno pretraživanje koristi posljednju točku izvan površine i trenutačnu točku za traženje dubinske vrijednosti.



Slika 34. Linearno pretraživanje prve točke presjeka

2.4.8 Samostalno bacanje sjene (engl. Self-shadowing)

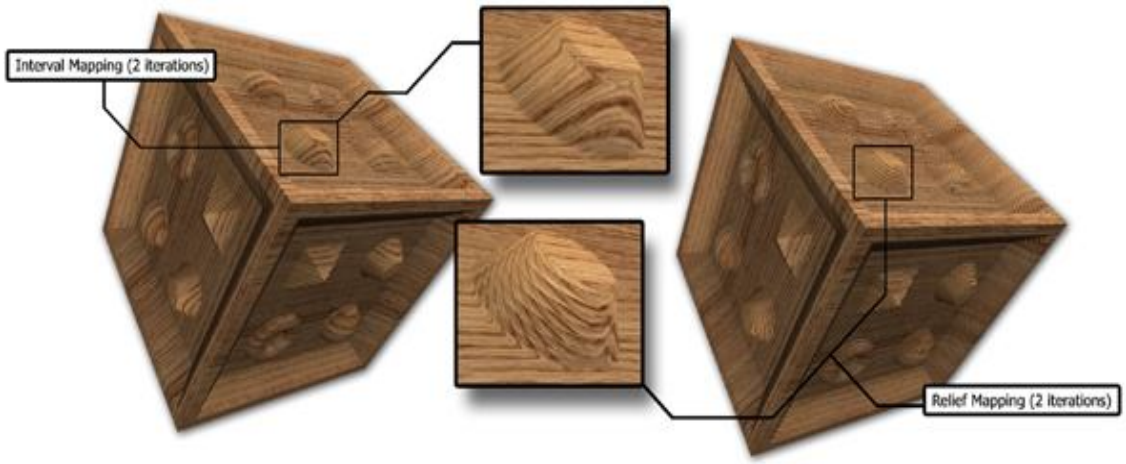
Algoritam za određivanje dali je trenutni fragment u sjeni ili ne je jednostavan i zasniva se na provjeri dali zraka svijetlosti sječe visinsku karakteristiku na putu prema točci koja se trenutno obrađuje. Ako da onda je slikovni element u sjeni, u suprotnom ne.



Slika 35. Ako zraka povučena iz točke C sječe dubinsku karakteristiku prije nego što sječe točku presjeka zrake gledanja i dubinske karakteristike, točka je u sjeni.

2.4.9 Intervalno preslikavanje

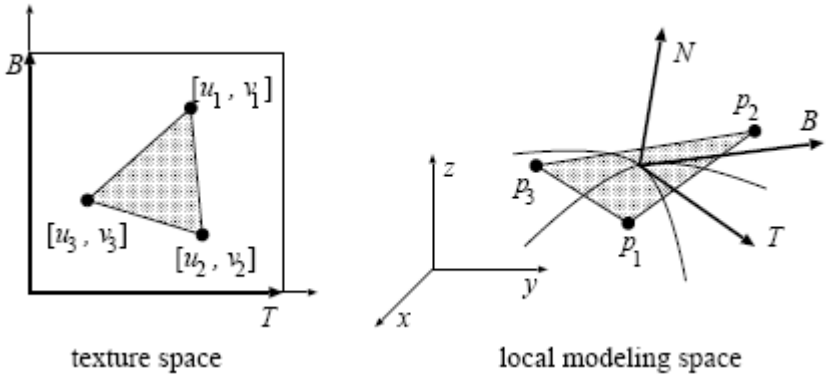
Poboljšanje binarne pretrage korištene u reljefnom preslikavanju tako da se kreira linija između poznatih unutrašnjih i vanjskih slikovnih elemenata te odabir sljedeće točke presijecanjem te linije sa zrakom, umjesto korištenja središnje točke.



Slika 36. Intervalno preslikavanje

2.5 Sustav tangente

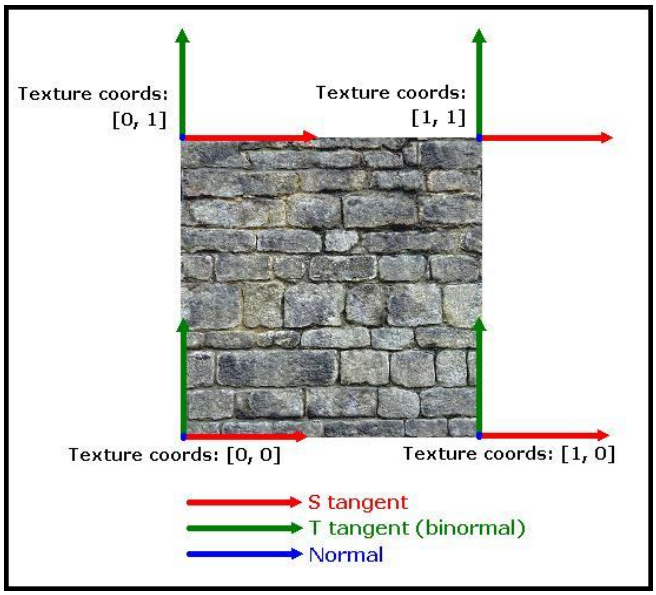
Vektori koji se koriste pri proračunu svijetla moraju biti definirani unutar istog koordinatnog sustava, taj sustav nije sustav koordinata oka (pozicija promatrača) iz razloga što bi količina operacija s normalama (prebacivanje, dodavanje pravoj normali i normalizacija) bila prevelika. Iz tog razloga koristi se sustav tangente koji varira nad svakim vrhom poligona. Svaki vrh ima svoj sustav tangente, a čine ga 3 međusobno okomita vektora, *Normala*, *Tangenta* i *Bi-Normala*. Tangentni prostor se koristi za tehnike koje rade perturbaciju normala.



Slika 37. Prikaz tangentnog prostora u koordinatnom prostoru teksture i 3D modeliranja. Binormala prati B vektor dok tangenta prati T vektor.

Preslikavanje iz lokalnog prostora objekta u prostor teksture radi se sljedećom formulom:

$$\begin{bmatrix} S_x \\ S_y \\ S_z \end{bmatrix} = \begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \\ N_x & N_y & N_z \end{bmatrix} \begin{bmatrix} O_x \\ O_y \\ O_z \end{bmatrix}$$



Slika 38. smjerova vektora u prostoru teksture

2.6 Programska implementacija reljefnih tehnika teksturiranja

2.6.1 Odabir Programske platforme

Prilikom odabira platforme za programsku implementaciju izbor se svodio na nekoliko mogućnosti:

- OpenGL:
    - prednosti-> otvoreni kod niske razine, velik broj stranica s uputom, rad s OpenGLom tijekom studija, pridruženi i dobro dokumentirani jezik za sjenčanje (GLSL)
    - mane-> konfliktne informacije, neke stranice s uputama pisane za druge operacijske sustave, nekompatibilnost GLUT biblioteke s VisualStudiom 2008
- DirectX:
    - prednosti-> većina današnjih igara se razvija na ovom API-ju, podrška za širok spektar efekata
    - mane-> nije otvorenog koda, neiskustvo u radu s istome, potrebno bi bilo učiti drugi jezik za sjenčanje (HLSL)
- Otvoreni graf Scene (engl. Open Scene Graph)
    - prednosti-> otvoreni kod, u konstantom razvoju, velik broj postojećih aplikacija, iskustvo u radu s njime tijekom studija
    - mane-> relativno kompliciran, velik broj nekompatibilnosti s računalnim sklopovljem i operacijskim sustavima, velik broj potrebnih dodatnih programskih dodataka (engl. plugin)

- Irrilicht grafički pokretač otvorenoga koda :
    - prednosti->otvoreni kod, neovisan o platformi i APIju (Direct3D, OpenGL software), velik broj podržanih formata i detaljne i dobro komentirane upute, objektno orijentiran
    - mane-> neiskustvo u radu s njime

Nakon isprobavanja navedenih opcija (razmatranja prednosti i mana, te pisanja jednostavnih implementacija u njima) odabir je spao na Irrilicht open source engine zbog njegove fleksibilnosti i aktivnih foruma koje čine prilagođavanje na njega relativno brzim i jednostavnim. Sama aplikacija koristi OpenGL sučelje za programiranje aplikacija (engl. *Application programing interface*) te programe za sjenčanje pisane u OpenGL jeziku za sjenčanje (GLSL).

2.6.2 OpenGL jezik za sjenčanje (OpenGL Shading Language)

Programi za sjenčanje su postali dostupni tek na karticama Nvidia Geforce 3 i Ati Radeon 8500 i novijim, sa sobom su donijeli veliki broj novih efekata obzirom da omogućavaju programeru da programira sam grafički cjevovod (engl. Graphics pipeline), umjesto da samo konfigurira isti kao na ranijem grafičkom sklopovlju.

OpenGl jezik za sjenčanje, također poznat pod nazivima GLSL ili GLslang, je jezik za sjenčanje visoke razine zasnovan na programskom jeziku C. Kreiran je od strane OpenGL odbora za ocjenjivanje arhitekture (engl. *OpenGL arhitecture review board*) kako bi pružio programerima direktnu kontrolu nad grafičkim cjevovodom, bez ulaženja u asemblerski kod. Neke od koristi koje GLslang donosi su kompatibilnost s različitim platformama (Windows, MacOS, Linux), mogućnost pisanja programa za sjenčanje na bilo kojoj grafičkoj kartici, podržava GLSL, svaki proizvođač grafičkih kartica uključuje GLSL kompajler u svojim upravljačkim programima omogućavajući kreaciju programa optimiziranih za arhitekturu pojedinačne grafičke kartice.

Sam OpenGL jezik za sjenčanje je relativno sličan programskim jezicima C i C++ s iznimkom korištenja pokazivača (engl. *pointers*), podržava petlje, grananje, logičke uvjete (if, else, while), te druge ugrađene funkcije, uz funkcije koje programeri sami mogu napisati. GLSL programi za sjenčanje nisu samostalne aplikacije, potrebna im je aplikacija koja koristi OpenGL API. Sami programi za sjenčanje su zapravo samo niz znakova (stringova) koji se prosljeđuju OpenGL upravljačkom programu. Programi za sjenčanje se mogu pisati unutar neke aplikacije ili tekstualnih datoteka, no moraju se slati kao niz znakova.

Postoje 3 glavne vrste programa za sjenčanje:

1. Programi za sjenčanje fragmenata proizvedenih rasterizacijom (engl. fragment shader)
2. Programi za sjenčanje vrhova (engl. *vertex shader*)
3. Programi za sjenčanje geometrije (engl. geometry shader)

Primjeri GLSL tehnika su višestruko preslikavanje teksture (engl. *multitexturing*), preslikavanje uz efekt paralakse, reljefno preslikavanje, cell-shading i razne druge.

Slika 39. Cell shading (lijevo),. preslikavanje okoline (engl. Enviroment mapping) (desno).

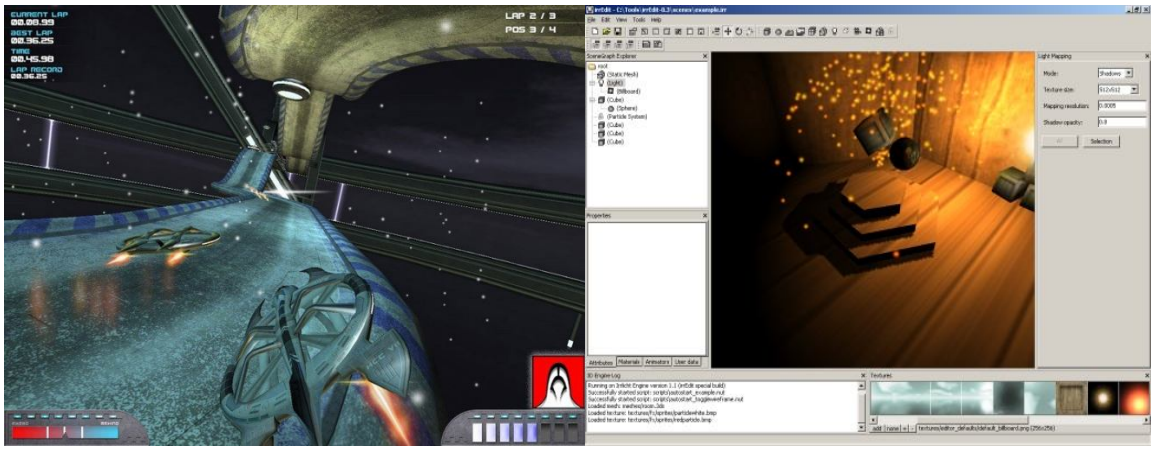
2.6.3 Irrilciht grafički pokretač

Ime za grafički pokretač otvorenog koda (engl. *Opensource graphics engine*) pisan u programsko jeziku C++, razvijen od strane Nikolaus Gebhardta. Podržava Windows, MacOS X, Linux, Windows CE operativne sustave te je portiran za Xbox, playstation portable, Symbian OS i Iphone platforme. Irrilicht je zasnovan na grafu scene, podržava velik broj formata datoteka (.3ds, .obj, .x, .bsp, …) čineći korištenje modela i objekata iz drugih programa jednostavnim. Kompatibilan je sa širokim spektrom novog i starog sklopovlja te podržava DirectX 8, DirectX 9, OpenGL i programske načine iscrtavanja. Trenutna verzija Irrilicht grafičkog pokretača je 1.7.1.

Karakteristike:

- Neovisnost o platformi
- Brzo iscrtavanje u realnom vremenu koristeći OpenGL i Direct3D API-je
- Proširiva baza tipova materijala s podrškom za programe za sjenčanje vrhova, točaka na ekranu i geometrije (engl. *Vertex, pixel, geometry shaders*)
- Glatki prijelazi između interijera i eksterijera zahvaljujući podesivom konfiguracijom scene
- Čestični efekti, panoi (engl. *billboard*), svjetlosne mape (engl. *lightmaps*), preslikavanje okoliša (engl. *enviroment mapping*) i mnogi drugi
- Skeletalna i morfirajuća animacija modela (engl. *skeletal & morph animation*)
- Nekoliko jezičnih poveznica (engl. *Language bindings*) koje omogućuju grafičkom pokretaču da bude dostupan i drugim jezicima kao C#, Java, VisualBasic, Delphi itd.
- Dva ugrađena programska načina iscrtavanja neovisna o platformi

- Jednostavan i fleksibilan sustav dvodimenzionalnog korisničkog sučelja
- Velik broj načina manipulacije dvodimenzionalnom grafikom i kombiniranje dvodimenzionalnih i trodimenzionalnih objekata
- Dobro dokumentirano sučelje za programiranje aplikacija (engl. Application programming interface, API) te velik broj primjera
- Pisan u C++ i u potpunosti objektno orijentiran
- Brza i laka detekcija kolizije te optimizirane knjižnične datoteke za trodimenzionalnu matematiku
- Direktno čitanje komprimiranih vrsta datoteka (.zip, .rar, .pak)
- Integrirani brzi XML parser



Slika 40. i 41. Primjeri nekih od mogućnosti Irrilicht grafičkog pokretača

2.6.4 Struktura programa

Struktura programa pisanih u Irrilicht grafičkom pokretaču otvorenog koda je slična klasičnoj C++ strukturi, deklariraju se funkcije koje se nakon toga koriste u main funkciji. Prvo je potrebno uključiti zaglavlje (engl. header) samog grafičkog pokretača i deklarirati prostor imena (engl. namespace) u kojem radimo, to se postiže sljedećim linijama koda:

#include <irrlicht.h>

using namespace irr;

Program sadrži nekoliko globalnih varijabli :

Deklaracija korijenskog Irrilicht objekta

```
IrrlichtDevice* device = 0;
```

Varijabla za odabir vrste programa za sijenčanje prilikom pokretanja programa

```
bool shaderchooser =false;
```

Varijabla za mijenjanje parametra preslikavanja uz efekt paralakse :

```
float parametar = 0.01;
```

Varijabla za mijenjanje parametra refljefnog preslikavanja:

```
float linear_search_steps = 11;
```

Nakon toga slijede deklaracije potrebnih funkcija, potrebna je funkcija koja obrađuje korisnički ulaz (engl user input):

```
class MyEventReceiver
```

I obzirom da se koriste programi za sjenčanje, funkcija koja im postavlja inicijalne podatke kako bi bili sposobni odrađivati daljnje izračune:

```
class MyShaderCallBack
```

Finalno slijedi deklaracija main funkcije kao u svakom C++ programu.

2.6.4.1 Metoda ponovnog poziva za postavljanje konstanti programa za sjenčanje (engl. Shader callback)

U ovoj implementaciji koriste se 2 različita programa za sjenčanje, prvi je jednostavna implementacija preslikavanja uz efekt paralakse, dok je drugi implementacija reljefnog preslikavanja. Obzirom da se mogu koristiti 2 različita programa za sjenčanje potrebno je predati specifične parametre programu za sjenčanje koji je odabran prilikom pokretanja programa. Ovo je moguće napraviti unutar jedne ShaderCallback metode ili s dvije zasebne metode, radi jednostavnosti sve se obavlja unutar `MyShaderCallBack` klase.

```
class MyShaderCallBack : public video::IShaderConstantSetCallBack
```

`IShaderConstantSetCallBack` je interno sučelje Irrilicht grafičkog pokretača koje je namijenjeno postavljanju konstanti za programe za sjenčanje za svaki iscrtani okvir. Klasak kojom postavljamo parametre mora naslijediti nju, potrebno je prepisati njezinu predefiniranu metodu *`virtual void OnSetConstants`* s vlastitom implementacijom.

```
virtual void OnSetConstants(video::IMaterialRendererServices*
                            services,s32 userData)
```

Metoda prima *`IMaterialRendererServices`* sučelje za manipulacijom odabranim upravljačkim programima i korisničke podatke, te se poziva svaki put kada je materijal koji koristi navedeni program za sjenčanje postavljen.

Unutar same metode parametri se postavljaju funkcijama:

```
virtual bool setPixelShaderConstant(const c8* name, const f32* floats,
                                    int count)
virtual bool setVertexShaderConstant(const c8* name, const f32* floats,
                                     int count)
```

Gdje *name* predstavlja ime uniformne varijable koju šaljemo programu za sjenčanje, *floats* predstavlja pokazivač na niz float vrijednosti a *count* predstavlja broj float vrijednosti u nizu. Primjerice za uniformnu varijablu tipa vec3 potrebno je poslati polje float brojeva od 3 elementa. Za podešavanje veličine teksture koristimo funkciju `smgr->getMeshManipulator()->makePlanarTextureMapping` čiji su parametri mreža poligona i faktor skaliranja.

2.6.4.2 Prijemnik događaja (engl. Event receiver)

Obzirom da je razvijena aplikacija s idejom interaktivnosti, potrebno je obrađivati ulaz od strane korisnika. Za to se koristi klasa `MyEventReceiver` derivirana iz klase public `IEventReceiver` koja je Irrilichtovo sučelje za objekte koji mogu procesirati događaje (engl. event). EventReciver može obrađivati korisnički unos od tipkovnice, miša, upravljačkih palica (engl. Joystick) i grafičkog sučelja (engl. graphical user interface) kao primjerice liste za selekciju i unost teksta.

Postoji jedna jedina metoda koju je potrebno implementirati :

```
virtual bool OnEvent(const SEvent& event)
```

U metodi bilježimo da li je tipka pritisnuta ili ne:

```
{
if (event.EventType == irr::EET_KEY_INPUT_EVENT)
KeyIsDown[event.KeyInput.Key] = event.KeyInput.PressedDown;
return false;
}
```

I dali je tipka kontinuirano pritisnuta:

```
virtual bool IsKeyDown(EKEY_CODE keyCode) const
{
return KeyIsDown[keyCode];
}
```

Te informacije se spremaju za svaku tipku u polje podataka, nakon toga unutar main petlje potrebno je tijekom kreacije Irrilicht korijenskog objekta (Irrilicht device) predati EventReciever kao parametar.

```
MyEventReceiver receiver; //dodaj prijamnik događaja
// kreiraj korijenski objekt
device = createDevice(driverType, core::dimension2d<u32>(1024, 768), 32,
false, false, false, &receiver);
```

Same korisničke akcije jednostavno obrađujemo unutar while petlje main funkcije:

```
if(receiver.IsKeyDown(irr::KEY_KEY_W)) {//kod za obradu}
```

2.6.4.3 Main funkcija

Glavna funkcija programa započinje tako da se odabere upravljački program koji ćemo koristiti. U ovom slučaju izbor je OpenGL, ako se želi korisniku dati izbor između različitih upravljačkih programa to je moguće na sljedeći način:

```
video::E_DRIVER_TYPE driverType=driverChoiceConsole();
if (driverType==video::EDT_COUNT)return 1;
```

U okviru ovog rada su programi za sjenčanje pisani samo u OPENGL jeziku za sjenčanje, iz tog razloga postavlja se OpenGL kao upravljački program.

```
video::E_DRIVER_TYPE driverType=video::EDT_OPENGL;
```

Prilikom pokretanja programa potrebno je pitati korisnika da li želi preslikavanje uz efekt paralakse ili reljefno preslikavanje teksture.

```
char i;
printf("Please press 'y' if you want to use relief mapping shaders.\n");
std::cin >> i;
if (i == 'y')shaderchooser = true;
```

Kao što je i prije opisano kreira se EventReciever i proslijedi kao parametar prilikom kreacije Irrilicht korijenskog objekta.

```
MyEventReceiver receiver; //dodaj event reciever
// create device
device = createDevice(driverType, core::dimension2d<u32>(1024, 768), 32,
false, false, false, &receiver);
```

Kreiramo objekte za pristup upravljačkom programu, menađeru scene i grafičkom sučelju, ovo je bitno za kasnije zadatke.

```
video::IVideoDriver* driver = device->getVideoDriver();
scene::ISceneManager* smgr = device->getSceneManager();
gui::IGUIEnvironment* gui = device->getGUIEnvironment();
```

Kreiramo grafičko okruženje koristeći netom definirane elemente:

```
gui->addImage(driver-
>getTexture("../../media/irrlichtlogo2.png"),core::position2d<s32>(10,10)
);
//postavljanje fonta
gui->getSkin()->setFont(gui->getFont("../../media/fontlucida.png"));
//text prozor s uputama
gui->addStaticText(L"tekst"core::rect<s32>(10,421,250,600), true, true,
0, -1, true);
```

Prije nego upotrijebimo programe za sjenčanje vrhova i fragmenata potrebno ih je prvo učitati ovisno o tome koju smo vrstu preslikavanja teksture odabrali.

```
//odabir relief ili parallax programa za sjencanje
if (shaderchooser)
{
```

```
psFileName = "../../media/AMfrag.frag";
vsFileName = "../../media/AMvert.vert";
}
else
{
psFileName = "../../media/fragment_shader.frag";
vsFileName = "../../media/vertex_shader.vert";
}
```

Radimo neobaveznu provjeru da li je upravljački program sposoban izvršavati napisane programe za sjenčanje, u slučaju nepodržanih opcija, postavljamo ime datoteke programa za sjenčanje na „null“.

```
if (!driver->queryFeature(video::EVDF_PIXEL_SHADER_1_1) &&
            !driver->queryFeature(video::EVDF_ARB_FRAGMENT_PROGRAM_1))
{
device->getLogger()->log("WARNING: Pixel shaders disabled "\"because of
missing driver/hardware support.");
psFileName = "";
}
```

Nakon što smo odabrali željene programe za sjenčanje, potrebno je kreirati novi tip materijala s njima. Irrilicht izvorno podržava razne tipove materijala (preslikavanje teksture, preslikavanje izbočina, prozirnost). Sam tip materijala je MaterialType 32 bitna vrijednost unutar SMaterial struct. Za to kreiramo pokazivač na IGPUProgrammingServices te pozivamo metodu addHighLevelShaderMaterialFromFile. Njeni parametri su imena programa za sjenčanje vrhova i fragmenata, ime i tip programa za sjenčanje, metoda za postavljanje konstanti i osnovni materijal.

```
video::IGPUProgrammingServices* gpu = driver>getGPUProgrammingServices();
s32 Materijal = 0;

if (gpu)
{
MyShaderCallBack* mc = new MyShaderCallBack();
//pozovi metodu za postavljanje konstanti programa za sjencanje
if (shaderchooser)
{

//postavi materijal ovisno o biranom tipu programa za sjencanje
Materijal = gpu->addHighLevelShaderMaterialFromFiles(
vsFileName, "vertexMain", video::EVST_VS_2_0,
psFileName, "pixelMain", video::EPST_PS_2_0,
mc, video::EMT_SOLID);
```

Za isprobavanje kreiranog novog materijala koristeći programe za sjenčanje potrebni su objekti u sceni. Irrilicht ima ugrađenu podršku za generaciju terena putem addTerrainMesh metode. Parametri koje dajemo su ime kreirane mreže (mesh-a), tekstura terena, visinska mapa za generaciju terena, veličina rastezanja, maksimalna visina, te razmak između vrhova (engl. dimension between vertices). Podržani su veliki tereni, teksture i visinska mapa mogu biti veliki do rezolucije od 8000 x 8000 slikovnih elemenata.

```
scene::IAnimatedMesh* terrainMesh = smgr->addTerrainMesh("teren",
      driver->createImageFromFile("../../media/terrain-texture.jpg"),
      driver->createImageFromFile("../../media/terrain-heightmap_2.bmp"),
      core::dimension2d< f32 >(10.0f, 10.0f),200.0f, core::dimension2d<
u32 >(64, 64));
```

Nakon što je kreiran teren, stvorimo čvor teren te mu dodajemo 2 mreže, prva je sam teren a drugi je tangentmesh obzirom da su potrebne tangente prilikom izračuna reljefnog preslikavanja. Nakon dodavanja mreže, postavljamo im materijal kreiran od strane programa za sjenčanje. Prvo postavljamo 2 teksture materijalu, prva je osnovna tekstura kakva bi se koristila pri preslikavanju teksture, druga je tekstura u kojoj je u RGB kanalima mapa normala, a u alfa kanalu visinska mapa. Postavljamo zastavicu EMF_LIGHTNING na true, jer želimo da površina reagira na osvijetljenje. Finalno postavljamo sam materijal, te mičemo mrežu tangente jer nam nije više potrebna.

```
scene::ISceneNode* teren = 0;
scene::IMesh* tangentMesh = smgr->getMeshManipulator()->
createMeshWithTangents(terrainMesh->getMesh(0));

teren = smgr->addMeshSceneNode(tangentMesh);
teren = smgr->addMeshSceneNode(terrainMesh);
teren ->setMaterialTexture(0, driver->
getTexture("../../media/wall.bmp"));
teren ->setMaterialTexture(1, driver->getTexture("../../media/grubi
prijelazi.bmp"));
teren ->setMaterialFlag(video::EMF_LIGHTING, true);
teren ->setMaterialType((video::E_MATERIAL_TYPE)Materijal);
tangentMesh->drop();
```

Kreiramo skybox radi boljeg izgleda okolnog 3D prostora, koristimo metodu `addSkyBoxSceneNode` čiji su parametri teksture (gornja, donja, lijeva, desna, prednja stražnja), te opcionalnog čvor roditelj i identifikator.

```
smgr->addSkyBoxSceneNode(
driver->getTexture("../../media/irrlicht2_up.jpg"),
driver->getTexture("../../media/irrlicht2_dn.jpg"),
driver->getTexture("../../media/irrlicht2_lf.jpg"),
driver->getTexture("../../media/irrlicht2_rt.jpg"),
driver->getTexture("../../media/irrlicht2_ft.jpg"),
driver->getTexture("../../media/irrlicht2_bk.jpg"));
```

Postavljamo kameru klasičnog tipa, miš za kontrolu pogleda, strelice na tipkovnici za kontrolu kretanja unutar scene. Obzirom da se kamera tretira kao čvor, moguće ju je pridružiti bilo kojem čvoru unutar grafa scene ako želimo da prati njegovo kretanje. U ovom slučaju kamera je slobodna, pozicija joj je postavljena relativno u odnosu na glavni čvor, te pogled usmjeren u centar koodrinatnog sustava, kursor miša je isključen.

```
scene::ICameraSceneNode* cam = smgr->addCameraSceneNodeFPS();
      cam->setPosition(core::vector3df(-100,50,10));
      cam->setTarget(core::vector3df(0,0,0));
```

```
device->getCursorControl()->setVisible(true);
```

Na kraju iscrtavamo scenu, sve dok je prozor aktivan, te obrađujemo korisnički unos, ovisno o tome koji tip programa za sjenčanje je odabran. Parametri koje mijenjamo su globalne varijable koje metoda za postavljanje konstanti predaje programu za sjenčanje.

```
while(device->run())
if (device->isWindowActive())
{
if (shaderchooser == false)
{
if(receiver.IsKeyDown(irr::KEY_KEY_W)) parametar = parametar  + 0.001;

if(receiver.IsKeyDown(irr::KEY_KEY_S))if (parametar > 0 )parametar =
parametar  - 0.001;

}

if (shaderchooser == true)
{
if(receiver.IsKeyDown(irr::KEY_KEY_A)) linear_search_steps =
linear_search_steps  + 1;

if(receiver.IsKeyDown(irr::KEY_KEY_D)) if (linear_search_steps > 0 )
linear_search_steps = linear_search_steps  - 1;

}
```

Počinjemo iscrtavanje scene metodom `beginScene`, parametri koje dajemo su uključen pozadinski međuspremnik (engl. backBuffer), uključen z međuspremnik (engl. Zbuffer) i boja za čišćenje pozadinskog međuspremnika.

```
driver->beginScene(true, true, video::SColor(255,0,0,0));
```

Iscrtavamo graf scene i grafičko sučelje, te signaliziramo upravljačkom programu da završi scenu.

```
smgr->drawAll(); //iscrtaj cijeli graf scene
gui->drawAll();   // iscrtaj kompletno graficko okruzenje
driver->endScene();
```

Nakon što je program gotov, pozivamo destruktor glavnog Irrilicht objekta.

```
device->drop();
```

2.7 Mjerenja

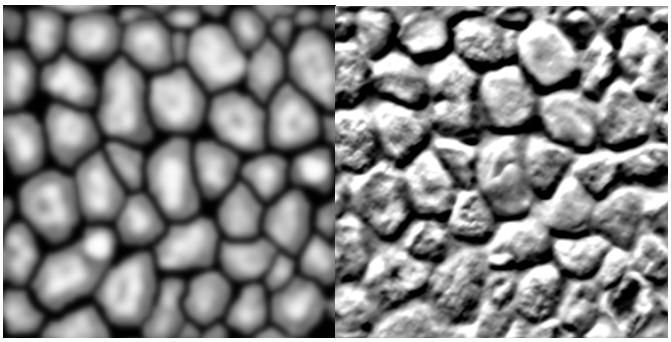
Testiranje je provedeno na 2 sustava, kao vrijednost broja iscrtanih okvira u sekundi uzimana je vrijednost prilikom pogleda na cjelokupni teren. Broj iscrtanih okvira po sekundi varira ovisno o smjeru gledanja promatrača. (Ako je manji dio terena u pogledu promatrača broj iscrtanih okvira će biti veći.)

Testna konfiguracija broj 1. :
Prijenosno računalo Acer Aspire 5020
Procesor : AMD Turion 1.6Ghz 64 bit
Radna memorija : 2GB DDR
Grafička kartica : Ati Mobility Radeon x700 (128 Mb vlastite memorije)
Hard Disk : 80GB ATA 100
Operativni sustav : Windows XP Service Pack 3

Testna konfiguracija broj 2 :
Procesor : AMD Athlon X2 5000+ (@2.7GHz) 64bit
Radna memorija : 2GB DDR 2 800 Mhz
Grafička kartica : Ati Radeon 4670 512 MB DDR3 RAM
Hard Disk : 640GB SATA2
Operativni sustav : Windows XP Service Pack 2

Testiranje je rađeno u „normalnim“ uvjetima, koji podrazumijevaju upaljen antivirusni program, vatrozid, te internetski pretraživač u pozadini. Na konfiguraciji broj 1. moguće je bilo testirati samo preslikavanje uz efekt paralakse, razlog tomu je što Mobility x700 grafička kartica sklopovski ne podržava instrukcije potrebne za izvođenje reljefnog preslikavanja. Preciznije, potrebna je verzija 2.0 OpenGLa a podržana verzija od grafičke kartice 1.5. Reljefno preslikavanje se pokreće i na prvoj konfiguraciji, no broj iscrtanih okvira po sekundi je maksimalno 1, uz 100% opterećenje procesora. U tim uvjetima testiranje nije bilo moguće provesti.

Obzirom da je za obje vrste preslikavanja potrebna visinska mapa i mapa normala, a visinske mapa gradova su generalno nedostupne, korištena je gruba aproksimacija visinske mape filtrom zrcalnog (spekularnog) osvjetljenja. Također je korištena jednostavna verzija preslikavanja uz efekt paralakse kojoj je dovoljna samo visinska mapa i glavna tekstura.



Slika 42. i 43. Visinska mapa i aproksimacija visinske mape zrcalnim (zrcalnim) osvjetljenjem

2.7.1 Preslikavanje uz efekt paralakse

Test 1. Urbani teren

Slika 44. Urbani Teren s preslikavanjem teksture

Slika 45. Urbani Teren s preslikavanjem uz efekt paralakse

Performanse :

Testna konfiguracija 1.	Broj iscrtanih okvira po sekundi
Preslikavanje teksture	45 FPS
Preslikavanje uz efekt paralakse	35 FPS

Testna konfiguracija 2.	Broj iscrtanih okvira po sekundi
Preslikavanje teksture	894 FPS
Preslikavanje uz efekt paralakse	867 FPS

Slika 46. Urbani teren promatran s veće udaljenosti

Test 2.

Urbana okolina niže rezolucije (proizvoljno odabrani fragment slike rezolucije 4096 x 4096)

Slika 47. Teren bez primjene preslikavanja uz efekt paralakse

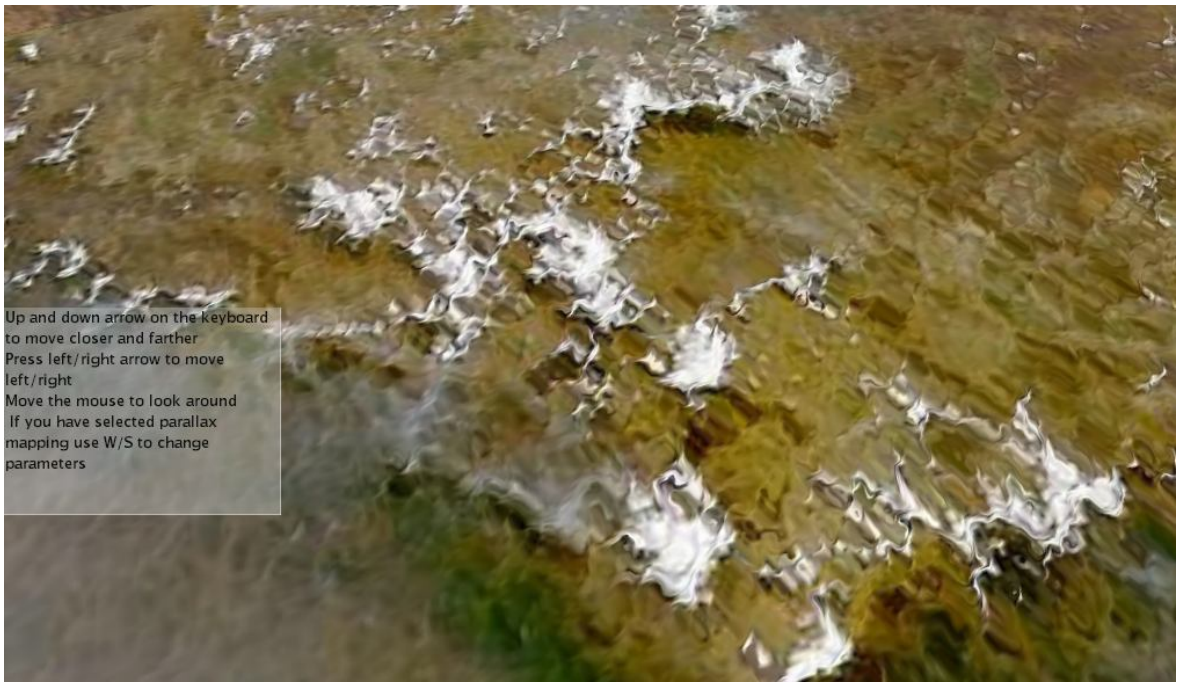
Slika 48. Teren sa primijenjenim preslikavanjem uz efekt paralakse

Testna konfiguracija 1.	Broj iscrtanih okvira u sekundi
Preslikavanje teksture	48 FPS
Preslikavanje uz efekt paralakse	36 FPS

Testna konfiguracija 2.	Broj iscrtanih okvira po sekundi
Preslikavanje teksture	885 FPS
Preslikavanje uz efekt paralakse	807 FPS

Test 3. Prirodni teren

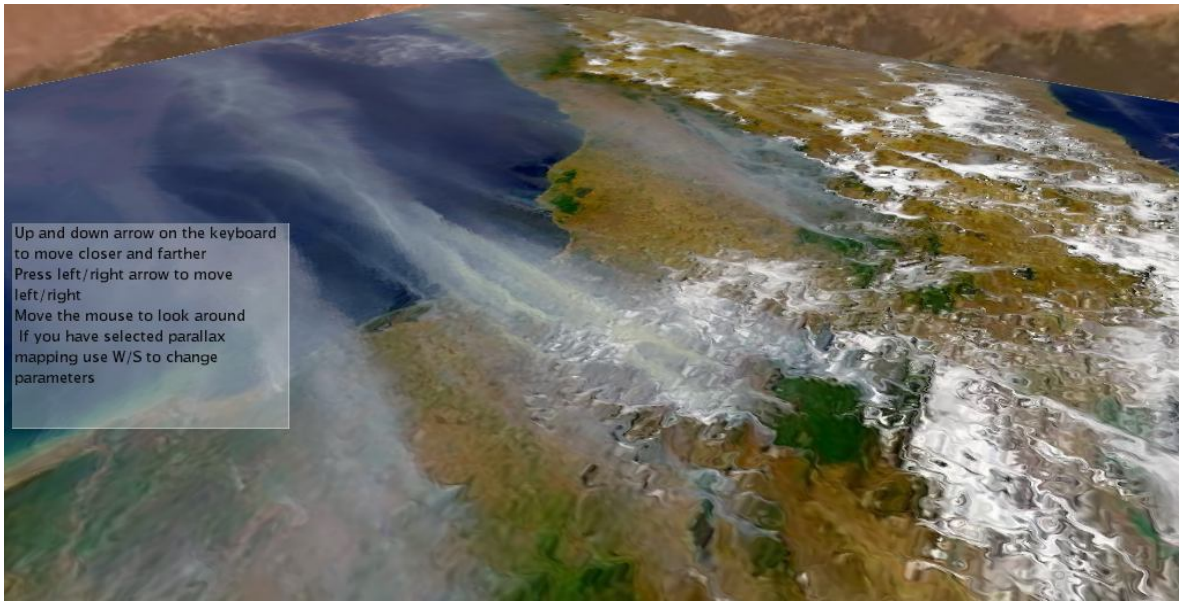
Slika 49. Prirodni teren s preslikavanjem teksture



Slika 50. Prirodni teren uz preslikavanje uz efekt paralakse

Testna konfiguracija 1.	Broj iscrtanih okvira u sekundi
Preslikavanje teksture	51 FPS
Preslikavanje uz efekt paralakse	43 FPS

Testna konfiguracija 2.	Broj iscrtanih okvira po sekundi
Preslikavanje teksture	893 FPS
Preslikavanje uz efekt paralakse	808 FPS



Slika 51. Pogled na teren s preslikavanjem uz efekt paralakse iz veće udaljenosti

Kao što se da vidjeti iz slika preslikavanje uz efekt paralakse daje dobru aproksimaciju trodimenzionalnosti dvodimenzionalne slike. Približavanjem teksturi dojam efekta opada u kvaliteti zbog aproksimativne prirode same tehnike. U kontekstu urbanih terena preslikavanje uz efekt paralakse može dati zadovoljavajući dojam trodimenzionalnosti i nepravilnosti objekata na površini uz mali utjecaj na performanse sustava. No preslikavanje uz efekt paralakse ne može u potpunosti zamijeniti 3D objekte kao dio simulacije, njegove mane su, kao što je već navedeno, isprekidanost i ispeglanost tekstura zbog aproksimacije odmaka, te deformacije i isticanje plošnosti kod malih kutova gledanja. Pri korištenju prirodnih terena, preslikavanje uz efekt paralakse je efikasna i jednostavna tehnika za podizanje vizualnog dojma i realizma scene uz mali trošak računalnih resursa.

2.7.2 Reljefno preslikavanje

Test 1. Urbani teren 1.:

Slično kao i kod testiranja preslikavanja uz efekt paralakse provedena su 3 testa, prvi je urbana tekstura veličine 512 x 512 slikovnih elemenata ponovljena po terenu, namjena ovog je provjera dali je reljefno preslikavanje u stanju dati zadovoljavajući prikaz velike urbane okoline.

Slika 52. Preslikavanje teksture na urbanom terenu

Slika 53. Reljefno preslikavanje na urbanom terenu

Testna konfiguracija 2.	Broj iscrtanih okvira po sekundi
Preslikavanje teksture	176 FPS
Reljefno preslikavanje	44 FPS

Test 2. urbani teren 2**.:**

U ovom testu dana je tekstura terena rezolucije 1024 x 1024 slikovnih elemenata te raširena preko cijelog terena. Namjena ovog testa je provjera kvalitete reljefnog preslikavanja pri bližim udaljenostima.

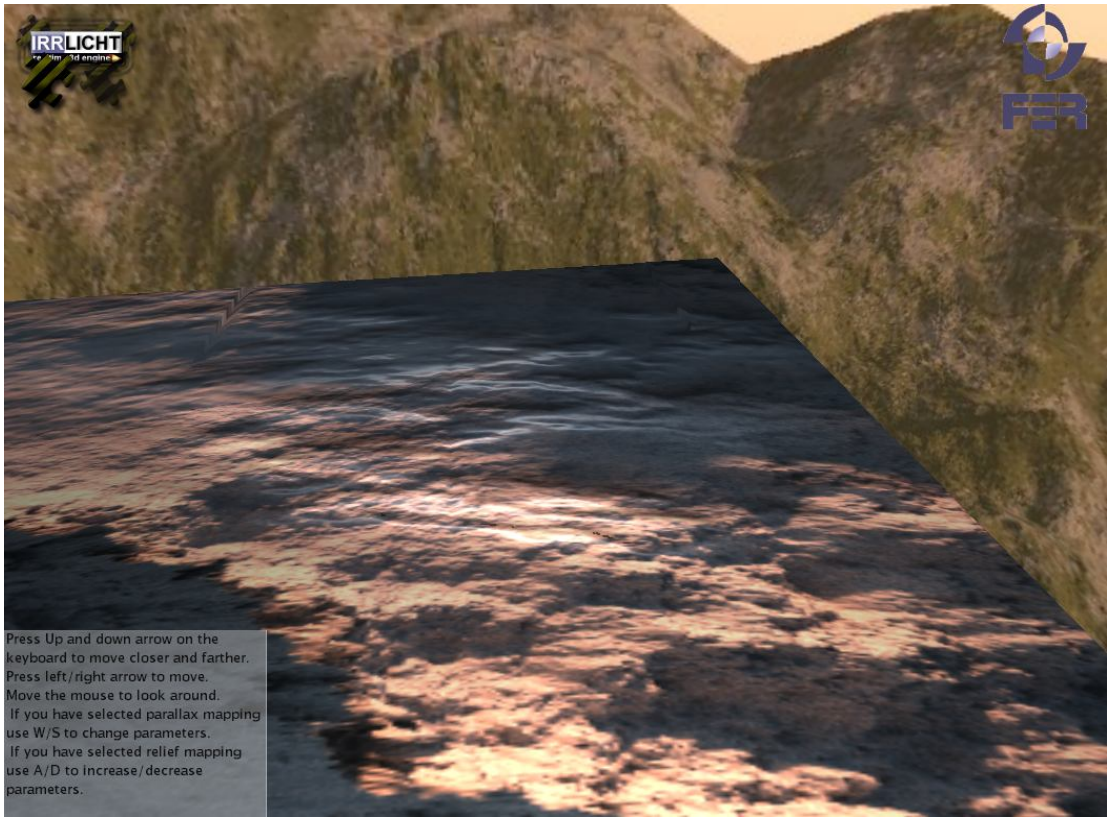
Slika 54. Urbani teren preslikavanje teksture

Slika 55. Urbani teren 2. Reljefno preslikavanje

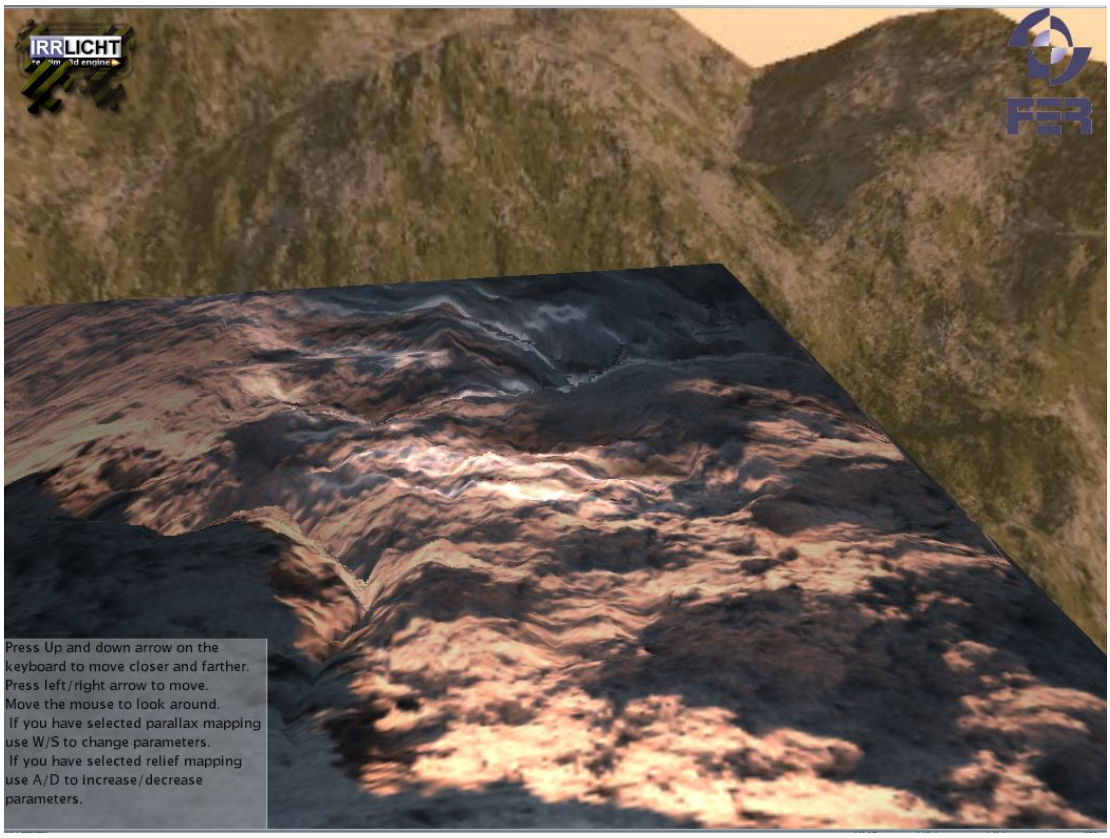
Testna konfiguracija 2.	Broj iscrtanih okvira po sekundi
Preslikavanje teksture	196 FPS
Reljefno preslikavanje	56 FPS

Test 3. Prirodni teren:

U ovom testu namjena je bila ustanoviti koliko realnu aproksimaciju grubog terena daje reljefno preslikavanje, korištena je tekstura terena rezolucije 1280 x 1280 slikovnih elemenata, s pripadajućom dubinskom karakteristikom i mapom normala.



Slika 56. Prirodni teren s preslikavanjem teksture



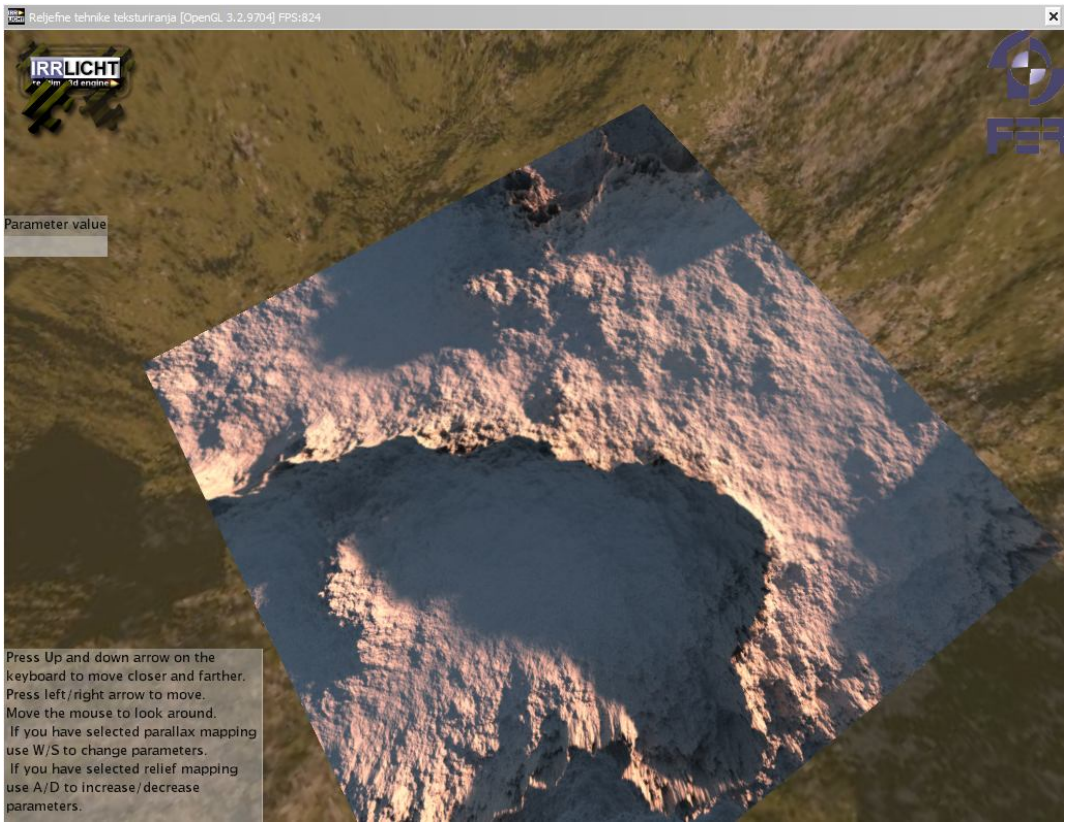
Slika 57. Prirodni teren s reljefnim preslikavanjem

Testna konfiguracija 2.	Broj iscrtanih okvira po sekundi
Preslikavanje teksture	170 FPS
Reljefno preslikavanje	60 FPS

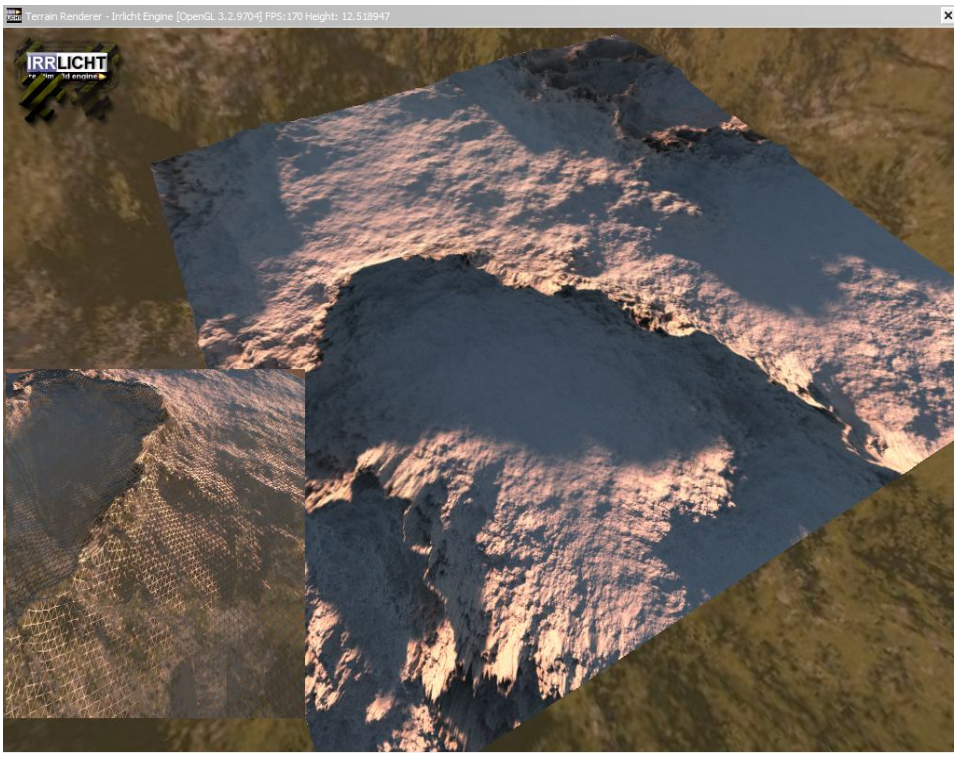
Test 4.: Usporedba terena generiranog iz teksture reljefnim preslikavanjem i terena generiranog poligonima

Irrilicht grafički pokretač interno podržava teren kao tip čvora, samo generiranje terena se radi pomoću `scene::ITerrainSceneNode* terrain = smgr->addTerrainSceneNode` funkcije. Mjerenje se provodi na teksturi veličine 1024 x 1024, parametri skaliranja teksture su 0.00035 za reljefno preslikavanje dok su parametri skaliranja terena za `addTerrainSceneNode` funkciju `(1.0f, 0.11f, 1.0f)`. Namjena ovog testa je usporedba brzine i kvalitete reljefnog preslikavanja naspram klasične metode iscrtavanja terena pomoću poligona. Također je napravljena usporedba sa preslikavanjem uz efekt paralakse, te usporedbe brzina različitih brzina iscrtavanja ovisno o broju razina detalja.

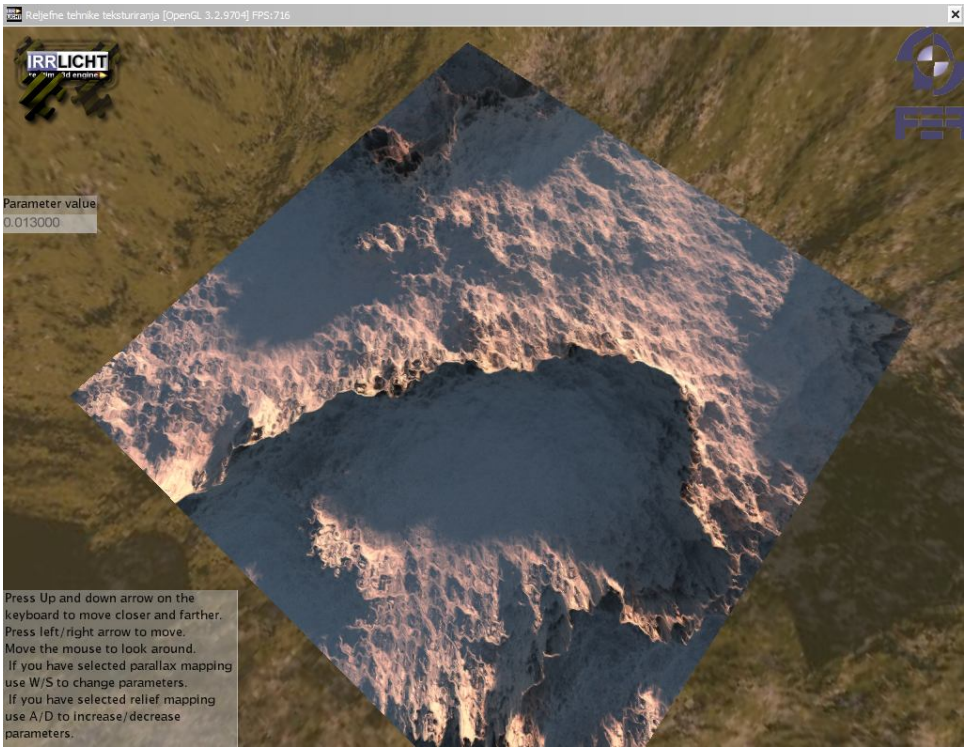
Testna konfiguracija 2.	Broj iscrtanih okvira po sekundi
Preslikavanje teksture	824 FPS
Prikaz terena poligonima (1. Razina detalja)	5 FPS
Prikaz terena poligonima (3. Razine detalja)	62 FPS
Prikaz terena poligonima (5. Razina detalja)	170 FPS
Preslikavanje uz efekt paralakse	716 FPS
Reljefno preslikavanje	55 FPS



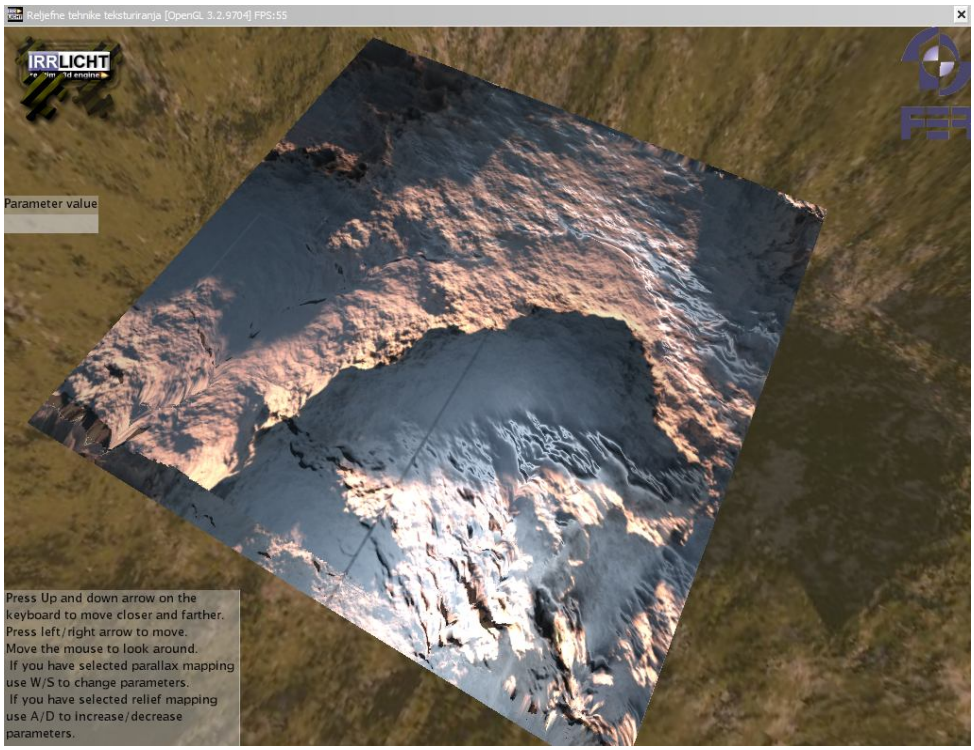
Slika 58. Prikaz terena teksturom



Slika 59. Prikaz terena poligonima



Slika 60. Prikaz terena preslikavanjem uz efekt paralakse



Slika 61. Prikaz terena reljefnim preslikavanjem

Kao što je vidljivo iz slika, reljefno preslikavanje daje dobre rezultate za srednje udaljenosti od terena, no obzirom da je to samo tehnika iluzije trodimenzionalnosti površine, ne bi bilo primjenjivo kod velike blizine (primjerice let aviona stotinjak metara od tla). Preslikavanje teksture je najbrža tehnika, no ne daje nikakav dojam trodimenzionalnosti. Klasično iscrtavanje terena pomoću poligona dalje veći broj iscrtanih okvira po sekundi, tome su najviše zaslužne funkcije za reguliranje razine detalja (engl. Level of Detail). Te funkcije dinamički smanjuju kompleksnost poligona koji se udaljava od promatrača, a povećavaju kompleksnost onih koji se približavaju promatraču. U tablici je vidljivo da je broj iscrtanih okvira po sekundi direktno povezan sa brojem razina detalja. U slučaju da se ne koriste razine detalja tehnika reljefnog preslikavanja bi bila daleko bolji izbor. Preslikavanje uz efekt paralakse s druge strane daje dobar dojam, no primjetna je deformacija teksture pogotovo pri manjoj udaljenosti od terena.

3. Zaključak

Tijekom ovog rada promatrali smo tehniku preslikavanja uz efekt paralakse i reljefnog preslikavanja, njihove teoretske osnove i njihovu primjenu u prikazu terena. Najveći problem prilikom izrade rada je bila nedostupnost visinskih mapa gradova. Iz tog razloga za preslikavanje uz efekt paralakse je korištena aproksimacija visinske mape pomoću zrcalnog osvjetljenja, dok su za reljefno preslikavanje korištene ručno kreirane visinske mape.

Iz provedenih mjerenja se vidi da je tehnika preslikavanja uz efekt paralakse, jednostavan i efikasan način simulacije neravnina na površini terena. Prosječan pad performansi pri korištenju preslikavanja uz efekt paralakse je 14.06%, što je računajući snagu modernih računala, neprimjetno obzirom na velik broj iscrtanih okvira po sekundi. Preslikavanje uz efekt paralakse koristi visinsku mapu i mapu normala, no aproksimacijom visinske mape sa zrcalnim osvjetljenjem moguće je brzo i jednostavno generirati neravnine na terenu. Ova tehnika je korisna na velikim i srednjim udaljenostima od promatranih objekata, no približavanjem objektima i gledanjem iz malih kutova njene mane dolaze do izražaja.

Reljefno preslikavanje je tehnika koja daje puno bolje vizualne rezultate od preslikavanja uz efekt paralakse, no uz znatno veći pad performansi. Prosječan pad performansi s uključenim reljefnim preslikavanjem je bio 69.75%, čineći ovu tehniku neefikasnom za konstantno korištenje u sceni. Reljefno preslikavanje daje dobru simulaciju zgrada u urbanim okolinama i neravnina na terenu kod prirodnih okoliša kod srednje i relativno bliske udaljenosti od promatrača. Prednost reljefnog preslikavanja je ta što ne dodaje nikakvu dodatnu geometriju, no kod primjene na samim terenima prikaz poligonima je bolja opcija zbog algoritama za regulaciju količine detalja (engl. Level of detail). Obzirom na kompleksnost tehnike zbog korištenja rekurzivne funkcije praćenja zrake, tehnika reljefnog preslikavanja bi bila najbolje iskorištena za određene dijelove terena. Primjerice, reljefno preslikavanje samo za urbane dijelove kada se približe na određenu udaljenost od promatrača. Ovo je jednostavno ostvariti tako da u visinskoj mapi dijelove koji nisu urbano područje sve označimo istom visinom. U slučaju većih udaljenosti kao primjerice let na nekoliko kilometara avionom bilo bi kontraproduktivno koristiti preslikavanje zbog činjenice da se trodimenzionalnost objekata neće primijetiti u velikoj mjeri (alternativa u tom slučaju bi bilo preslikavanje uz efekt paralakse).

Sve tehnike spomenute u radu su tehnike koje se aktivno koriste u računalnoj industriji, pogotovo industriji igara za povećavanje realnosti iscrtanih scena. Reljefno preslikavanje je tehnika koja će se vjerojatno sve više koristiti u budućnosti obzirom da se konstantno traže načini poboljšavanja iste, kao primjerice reljefno preslikavanje optimizirano mapom stožaca.

4. Literatura

1. Nedić, Anamarija, „Reljefne tehnike teksturiranja“ Fakultet Elektrotehnike i računarstva, Zagreb, Dostupno na http://crosbi.znanstvenici.hr/prikazi-rad?&lang=EN&rad=431885, 2009.
2. Fabio Policarpo, Manuel M. Oliveira Comba, Joao „Real-Time Relief Mapping on Arbitrary Polygonal Surfaces“. April 3-6, 2005.
3. Zink, Jason, „A closer Look At Parallax Occlusion Mapping.” Dostupno na http://www.gamedev.net/columns/hardcore/pom/, 2006.
4. http://en.wikipedia.org/wiki/Bump_mapping
5. http://en.wikipedia.org/wiki/Heightmap
6. http://en.wikipedia.org/wiki/Parallax_mapping
7. http://en.wikipedia.org/wiki/Relief_mapping_%28computer_graphics%29
8. Eric Risser, Musawir Shah i Sumanta Pattanaik, „Interval Mapping: Quick Per-pixel Displacement Mapping.“ University of Central Florida, Dostupno na http://graphics.cs.ucf.edu/IntervalMapping/, 2006.
9. „Irrilicht open source engine“, Dostupno na http://irrlicht.sourceforge.net/tutorials.html, 2010.
10. Jouvie ‚Jérôme, „OpenGL - Lesson 8 : Tangent Space“, Dostupno na http://jerome.jouvie.free.fr/OpenGl/Lessons/Lesson8.php, 2008.
11. http://en.wikipedia.org/wiki/GLSL
12. Welsh , T. „Parallax Mapping with Offset Limiting: A Per-Pixel Aproximation of Uneven Surfaces“, Infiscape Corporation, January 2004.
13. Policarpo, Fabio, i Oliveira, Manuel M. „Relief Mapping of Non-Height-Field Surface Details.” In Proceedings of the 2006 Symposium on Interactive 3DGraphics and Games, pp. 55–62, 2006.
14. Jouvie ‚Jérôme, „Project 4 : Shaders“, Dostupno na http://jerome.jouvie.free.fr/OpenGl/Projects/Shaders.php, 2009.
15. Paul, „Simple OpenGL Bumpmapping tutorial“ Dostupno na http://www.paulsprojects.net/tutorials/simplebump/simplebump.html

5. Sažetak

(Reljefne tehnike teksturiranja u prikazu terena)

U radu je dan pregled modernih tehnika simulacije trodimenzionalnosti površine pomoću teksture. Opisana je njihova teorijska i matematička podloga te njihova primjena u simulaciji urbanih i prirodnih terena. Napravljena je programska implementacija preslikavanja uz efekt paralakse i reljefnog preslikavanja u Irrilicht grafičkom pokretaču. Korišteni su programi za sjenčanje napravljeni pomoću GLSL jezika za sjenčanje te programa AMD RenderMonkey. Obavljena su mjerenja implementiranih tehnika na primjerima urbanih i prirodnih terena.

Ključne riječi: reljefno preslikavanje teksture, preslikavanje uz efekt paralakse, iscrtavanje terena, urbani teren, prirodni teren, iscrtavanje pomoću promjene uzoraka teksture, Irrilicht grafički pokretač, algoritmi praćenja zrake.

Abstract

(Relief mapping application in terrain rendering)

This paper presents and overview of some of the more known techniques for simulating the three-dimensionality of a flat surface. Their theoretical and mathematical bases are given, as well as their application in simulating urban and natural environments. A program is given which includes the implementations of parallax and relief mapping within the Irrilicht open source graphics engine. The shaders used were made with the GLSL shader language and the AMD RenderMonkey program. Tests were made to determine the possible usage of parallax and relief mapping in the display of natural and urban environments.

Keywords: relief mapping, texture mapping, parallax mapping, terrain rendering, urban terrain, natural terrain , Irrilicht open source engine, ray-tracing.