Drugi domaći

RAF - Računarska Grafika 2019/2020

Generalne informacije i zadatak

Cilj domaćeg zadatka je napraviti interaktivnu 3D aplikaciju. Domaći se piše u C-u (dozvoljen je i C++ ali nije preporučen bez predašnjeg iskustva) sa pomoćnim bibliotekama. Preporučeno je, ali ne i obavezno, korišćenje biblioteke RAFGL sa vežbi.

Ako se ne koristi biblioteka sa časa lista dozvoljenih biblioteka je:

- GLFW ili SDL
- Bilo koja OpenGL Loading biblioteka (kao što su GLEW, glad, GL3W, itd.)
- stb_image

Druge biblioteke dolaze u obzir, ali se obavezno konsultovati sa asistentom pre početka rada. Obavezno je koristiti OpenGL 3.3 verziju ili novije verzije. High level engine-i kao sto su Unity, Unreal, Ogre, itd. ne dolaze u obzir.

Zahtevi i pravila

- **Interaktivna aplikacija** - tema nije strogo definisana, slično kao i na prvom domaćem (osim kod specijalnih tema, videti ispod) i treba se napraviti interaktivna grafička aplikacija neke vrste, ovaj put u 3D prostoru. U ovom zadatku je i dalje fokus na tehničkoj realizaciji grafike i samo se to boduje.
- **OpenGL** - Koristiti OpenGL 3.3 ili novije, tj GLSL 3.30 ili novije
- **RAFGL** - Preporučeno je ali ne i obavezno korišćenje biblioteke *rafgl* sa vežbi. Nju možete i da editujete (pratite šta menjate radi lakše odbrane) ili da je ne koristite uopšte
- Domaći treba da sadrži barem:
    - Nove i/ili modifikovane šejdere (shader programe) u odnosu na ono što je rađeno na času
    - Proceduralno generisana i/ili dinamička geometrija (nešto što nije iz OBJ fajla)
    - Hijerarhije objekata gde se neki delovi programski kreću ili animiraju (točkovi, propeleri, ...)
    - Korišćenje drugih tekstura pored albedo (colour) tekstura
- **Zvuk je potpuno opcion** - OpenAL je preporuka ako radite zvuk (https://ffainelli.github.io/openal-example/)

Saveti

- Upoznajte sa postojećim kodom (strukture, funkcije i shaderi) preko primera sa vežbi
- **Logika aplikacije treba da bude jednostavna** - logika preseka, sudaranja, orijentacija objekata zna biti komplikovana, pa je treba i svesti na minimum ili se ograničiti na par tipova interakcija i par ravni
- **U logici koristiti aproksimacije objekata** - u logici kao što je detekcija pogotka, selekcija pomoću miša i ostaloj, aproksimirati objete jednostavnim oblicima kao što su lopte, kocke, cilindri kako bi matematika bila što jednostavnija
- **Real time rendering** - vodite računa o stvarima kao što su prekompleksni modeli (previliki broj poligona), nepotrebne računice, nepotrebne ili česte alokacije memorije, teške matematičke funkcije mnogo puta ponovljene, itd.

Specijalne teme

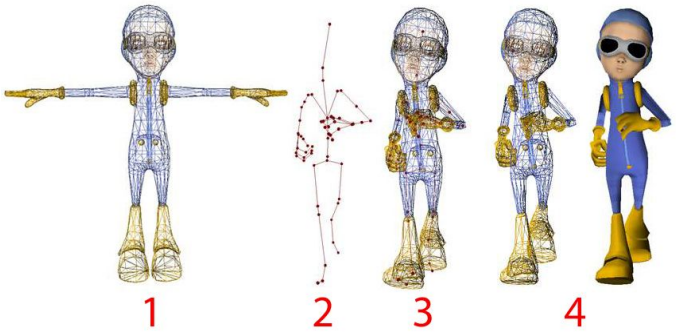
Specijalne teme su namenjene studentima koji žele strožije definisan projekat sa glavnom temom

Skeletalna animacija, IK

Implementirati sistem koji podržava skeletalnu animaciju kroz skinning u vertex shader-u. Skeletalna animacija koristi stablo objekata koje formira skelet, gde je jedna glavna (root) kost početna za dati objetak, a sve ostale su "nakačene" i zavise od nje. U ovom sistemu kosti nemaju zasebne geometrije, već sve zajedno utiču na istu geometriju i deformišu je, ali prema zadatim težinama (matrica za svaki vertex i svaku kost). Na osnovu matrice težina i stanja kostiju (zglobova) se u vertex shaderu računa težinska suma transformacionih matrica za te kosti i dobija se transformacija za taj jedan konkretan vertex (ovo se radi za sve vertexe).

Najteži deo ovog projekta je verovatno parsiranje fajlova u kojima su zapisani ovako uređeni modeli i animacije i zbog ovoga su dozvoljena ili custom rešenja, ili gotove bilbioteke (uz dogovor sa asistentom) ili proceduralne animacije.

Kada sistem profunkcioniše potrebno je demonstrirati njegovu dinamičnost nekom real time izmenom poze i/ili animacije, kao što su okretanje glave lika da prati neki proizvoljan objekat u svetu, korekcija zglobova kolena i stopala kako bi karakter bio u kontaktu sa podom u svim trenutcima (inverzna kinematika)



Skeletalna animacija

Inverzna kinematika

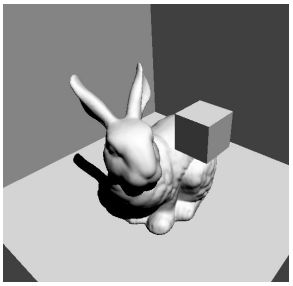
Korisni linkovi:

- https://moddb.fandom.com/wiki/OpenGL:Tutorials:Basic_Bones_System
- https://www.3dgep.com/gpu-skinning-of-md5-models-in-opengl-and-cg/
- https://www.youtube.com/watch?v=f3Cr8Yx3GGA (tutorijal je za Java jezik, ali je svakako dobro objašnjeno nezavisno od implementacije)

Shadow mapping, kaskade

Shadow mapping je danas sveprisutna tehnika za iscrtavanje „pravih“ senki, projektovanih i dinamičkih. Podrazumeva specijalno iscrtavanje scene iz perspektive svetla, pri čemu pamtimo samo depth (Z) vrednosti, tako da iz te perspektive znamo koliko je od njega daleko svaki piksel koji će to svetlo da dotakne. Zatim, znajući perspektivu svetla i položaj našeg pogleda, za svaki piksel na sceni, projektujemo njegovu poziciju u prethodno iscrtanu sliku iz ugla svjetla i poredimo da li je ta nova daljina veća ili manja od upisane – ako je veća, piksel je u senci, jer znači da na toj pravoj ka svetlu postoji nešto što je bliže svetlu i time baca senku na našu trenutnu poziciju, u suprotnom, piksel je osvetljen i radimo regularni shading.

Razmotriti probleme biasa/akni i filtriranja, zatim implementirati kaskade. Cascaded Shadow Maps (CSM) je optimizacija u kojoj se iscrtavaju dve ili više shadow mapa, koje zadržavaju rezoluciju, ali pokrivaju sve veći deo scene. Problem sa početnom implementacijom je što, zavisno od položaja kamere i položaja i ugla svjetla, može doći do toga da dosta rezolucije shadow mape bude bačeno na delove scene koji su daleko od kamere, dok će oni bliži izgledati mutno i/ili stepenasto zbog nedostatka rezolucije. Kaskade ovo pokušavaju da balansiraju, tako što će se prva mapa crtati tako da pokriva relativno mali deo scene, oko posmatrača, kako bi senke u tom delu bile kvalitetne, dok se nakon toga crta barem još jedna ili više njih, koje pokrivaju sve veće segmente scene (ali zadržavaju rezoluciju, tako da gustina piksela opada). Pri konačnom crtanju se na osnovu položaja tačke i položaja posmatrača i rasporeda kaskada bira prava mapa/kaskada za čitanje.

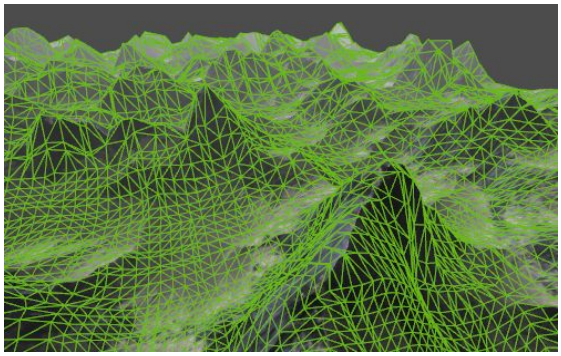


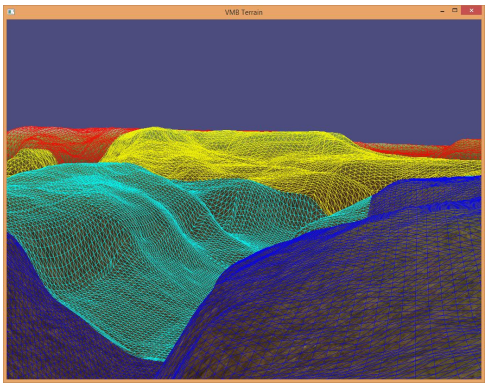
Korisni linkvoi:

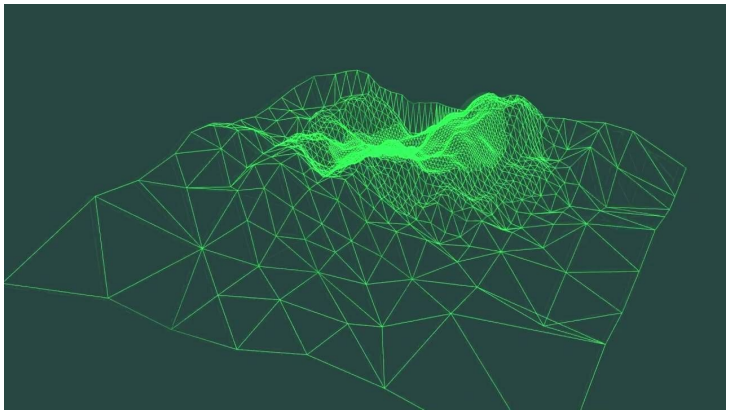
- https://en.wikipedia.org/wiki/Shadow_mapping
- http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping/
- http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.193.5620&rep=rep1&type=pdf
- https://ndotl.wordpress.com/2014/12/19/notes-on-shadow-bias/
- https://docs.microsoft.com/en-us/windows/win32/dxtecharts/cascaded-shadow-maps?redirectedfrom=MSDN (direct3D implementacija, ne OpenGL, svakako korsino)

Heightmap teren, LOD i teksture

Vrlo brz i efikasan način da se modeluju veliki otvoreni prostori je da im se reljef predstavi u obliku heightmape, tj. teksture/matrice, u kojoj svaki piksel/polje predstavlja visinu geometrije za tu poziciju. Od ovakvih podataka se lako dobija 3D model, jer možemo početi od savršeno ravne mreže kvadrata (po 2 trougla svaki), kojoj onda za svaki verteks upišemo vertikalnu koordinatu na osnovu pročitane vrednosti iz heightmape. Ovde treba voditi računa i o normalama (koje se mogu izračunati na osnovu uglova okolnih verteksa), kao i o UV koordinatama. Takođe, često je korisno implementirati i level-of-detail (LOD) sistem za ovakve terene, gde neće kompletan reljef biti generisan kao jedan mesh, već će biti rasparčan u blokove fiksne veličine. Za početak, ovako već možemo slati na crtanje samo one blokove koji dolaze u obzir da budu vidljivi, na osnovu položaja kamere i njenog ugla pogleda. Drugo, možemo lako napraviti više nivoa detaljnosti ovih blokova, praveći progresivno mesheve sa duplo manjom gustinom geometrije (npr. udvostručimo razmake, a za visinu uzimamo prosek od 2x2 okoline iz heightmape). Kada za svaki blok imamo nekoliko nivoa detaljnosti, biramo pravi na osnovu daljine tog bloka od kamere, ili još bolje, na osnovu procene koliki će procenat field-of-viewa kamere taj objekat da pokrije (za slučaj kamera koje mogu da zumiraju, gde će onda daljina da ostane ista, ali bi objekat trebao postati detaljniji). Razmotriti probleme sa ivicama između blokova različite detaljnosti i napraviti sistem koji će da proceduralno primeni i miksa teksture za ovakav teren, zavnisno od visine i nagiba svakog verteksa. Ovde bih preporučio fokus na sam osnovni koncept, pošto nije mnogo složen verovatno ga možete i sami implementirati ako ste razumeli opis, a reference možete iskoristiti za dodatne ideje.





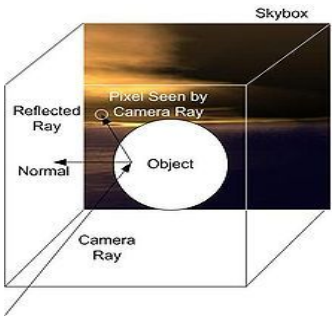


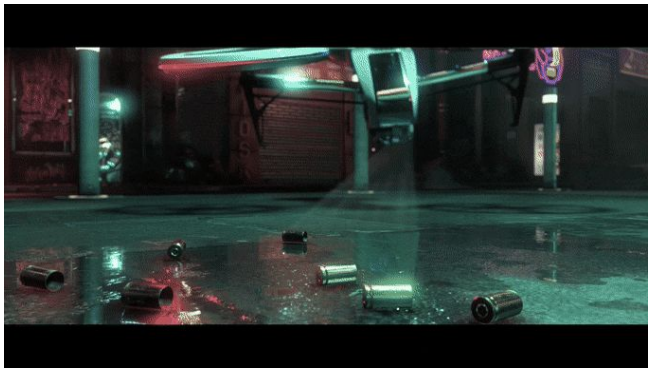
Korisni linkovi:

- https://blogs.igalia.com/itoral/2016/10/13/opengl-terrain-renderer-rendering-the-terrain-mesh/
- http://www.learnopengles.com/tag/height-maps/
- http://www.mbsoftworks.sk/tutorials/opengl4/018-heightmap-pt3-multiple-layers/

Real-time refleksije, enviroment probes

Odsjaje u realtime 3D grafici većinom postižemo koristeći environment mapu, obično u obliku cubemap, kao što ćemo/smo videli i na vežbama. Ovo je lako za razumeti i implementirati, dosta je i efikasno i izgleda uverljivo – dokle god se ništa u sceni ne pomera. Pošto je mapa refleksije statična i učitana iz fajla, to znači da nećemo u odsjaju na jednom objektu moći videti druge objekte, te da će i ako se taj prvi objekat pomeri predaleko od svoje početne pozicije odsjaji postati vidljivo pogrešni, jer će reflektovati neko okruženje u kom se ne nalazi više. Rešenje za ovo je da tokom rada programa dinamički ažuriramo tu cubemapu, renderujući u teksturu (FBO, kao na primerima sa vežbi). Ovaj posao bi mogao vrlo brzo da se nagomila, jer podrazumeva višestruko iscrtavanje scene po svakoj nezavisnoj refleksiji, zbog čega se u praksi broj objekata koji imaju ovakve refleksije dosta ograničava, najčešće na samo jedan. Obično samo jedan „glavni“ i najupadljiviji objekat ima korektne i ažurne refleksije, kao npr. lik ili vozilo igrača u igri. Ostalim objektima se refleksije ili ažuriraju mnogo ređe, ili recikliraju istu mapu koju koristi i glavni objekat, ili koriste neku od unaprijed definisanih mapa pripremljenih za razne tačke u sceni, zavisno od toga kojoj su najbliži. Sastaviti sistem kamere i renderovanja u teksturu tako da se za proizvoljnu tačku u prostoru scene može lako iscrtati cubemapa. Ovakvu mapu treba biti moguće upotrebiti za realtime refleksiju tokom rada, ili za čuvanje iscrtane mape na disk, kako bi se mogla upotrebiti kao statička mapa u narednom pokretanju. Sveukupno, u sistemu treba da bude moguće definisati nekoliko tačaka u sceni za koje će se automatizovano iscrtati i sačuvati statičke mape. A pri regularnom radu, za jedan objekat raditi stalno ažuriranje, dok za ostale refleksivne objekte pronalaziti najbližu prethodno definisanu tačku i uzimati refleksiju iz te pozicije, kako bi izgledala najmanje pogrešno.



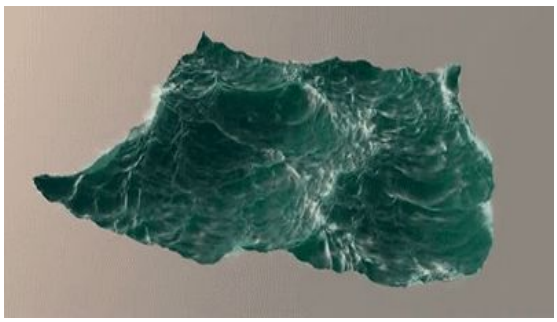
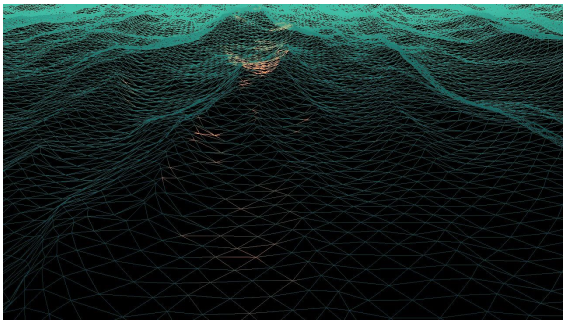
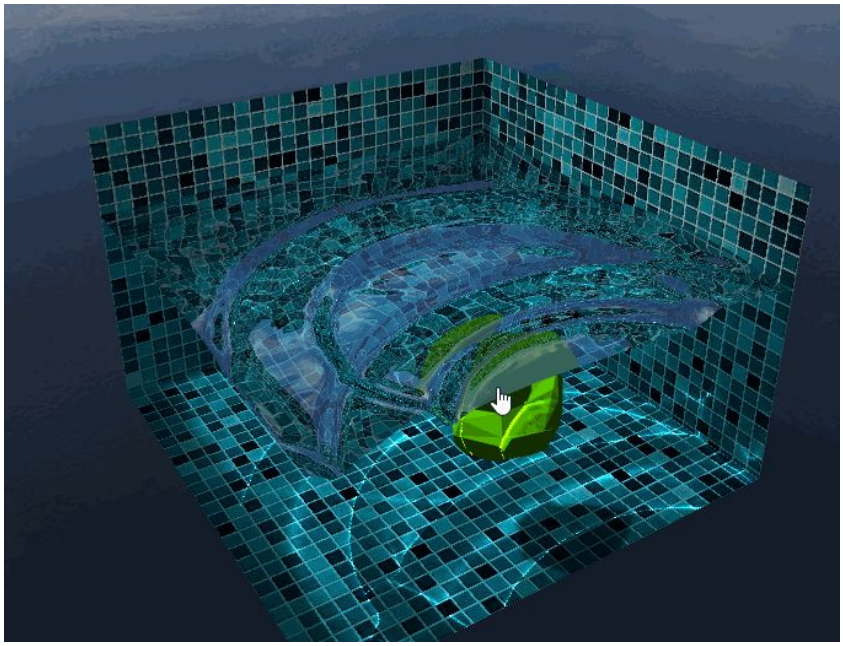


Korisni linkovi:

- https://learnopengl.com/Advanced-OpenGL/Cubemaps
- https://darrensweeney.net/2016/10/03/dynamic-cube-mapping-in-opengl/
- https://www.youtube.com/watch?v=lW_iqrtJORc(tutorijal je za Java jezik, ali je svakako dobro objašnjeno nezavisno od implementacije)
- http://webglsamples.org/dynamic-cubemap/dynamic-cubemap.html (WebGL demo)

Renderovanje vode, senčenje površine i deformacija

Voda je jedna od zanimljivijih stvari za iscrtavanje u 3D grafici, jer postoji dosta različitih i usko specijalizovanih pristupa, zavisno od toga kakve zahteve imamo za tu vodu – da li će voda biti mirna i površina ravna, da li je potrebno da se talasa (obično na neki vrlo ograničen način), da li je u pitanju samo pljusak/kapljice, ili je možda potrebno potpuno dinamičko kretanje fluida. Ovde ćemo uzeti slučaj gde će voda biti deformisana ravan, kao što bi slučaj bio sa nešto većim zapreminama vode, za koje uzimamo da se ne mogu uzburkati toliko da drastično promene oblik površine. Možemo ciljati na dva različita efekta: prvi bi bio onaj u kom je površina vode neinteraktivna, ali se kontstantno animira/talasa/deformiše po nekoj matematičkoj formuli (ili koristeći teksture); dok bi drugi bio onaj sa mirnijom vodom, barem u početku, koja je onda interaktivna, tako da možemo klikovima miša ili na neku drugu interakciju uzburkati površinu, pokreirajući talase koji će animirano da se propagiraju i deformišu površinu vode. U oba slučaja bismo počeli od ravni koja je sastavljena iz matrice kvadrata, tako da imamo dosta verteksa koje možemo da pomeramo oko njihovog početnog položaja da stvorimo talase, bilo da su interaktivni ili ne. Za površinu poput vode, osim oblika, jako je bitno i konačno sjenčenje, gde se implementiraju efekti poput releksije i refrakcije (po Fresnelu), uz možda „maglu“ za dubinu, penu na površini i slično.

Korisni linkovi:

- http://madebyevan.com/webgl-water/
- https://www.youtube.com/watch?v=HusvGeEDU_U
- http://matthias-mueller-fischer.ch/talks/GDC2008.pdf

Screen-space ambient occlusion (SSAO)

Jedna od najtežih prepreka na putu ka foto-realizmu u real-time renderovanju je simulacija indirektnog osvetljenja, odnosno, globalne interakcije svetla na sceni, gde se fotoni mogu mnogo puta odbiti od objekte, prenoseći obojenu svetlost, dok ne stignu do oka/kamere. Offline rendereri (poput Cycles u Blenderu) ovo simuliraju na (većinom) fizički korektan način, ali to ima svoju veliku cenu u performansama. U realnom vremenu taj pristup ne dolazi u obzir jer su potrebne brzine i za par redova veličine veće od onoga što imamo u takvom rendereru. Stoga se ovaj složen fenomen obično razlaže na jednostavnije aproksimacije, koje se onda nezavisno rade na različite načine. Jedna komponenta ovoga je aproksimacija ambijentalnog osvjetljenja, odnosno, koliko indirektne svetlosti stiže do svake date tačke na sceni. Doprinos ovog elementa u stvarnom životu najbolje možemo videti oblačnim danima – tada je skoro svo svetlo oko nas indirektno, raspršeno i odbijeno mnogo puta, tako da ništa više nema jasnu oštru senku, ali neke površine su ipak tamnije od ostalih, kao npr. put ispod automobila, razne rupe i udubljenja i generalno površine koje su u takvom položaju da su zaklonjene i od većine indirektnog osvetljenja. Korektno izračunavanje ovoga bi zahtevalo da za svaki piksel unutar fragment shadera imamo lako dostupne informacije o okolnoj 3D geometriji i brz način da tu okolinu „opipamo“ i proverimo koliko smo zaklonjeni od ambijentalne svetlosti. Ovo prosto nije izvodljivo na efikasan način sa današnjom arhitekturom grafike i grafičkih procesora; u toj fazi iscrtavanja nemamo nikakve informacije o geometriji okoline, niti način da je ispitujemo. Ono što možemo da uradimo je da vršimo aproksimaciju u dve (i po) dimenzije. Crtajući sliku, pazimo da imamo pristup depth (Z) bufferu na kraju, koji ćemo onda tretirati kao height-mapu (kao u temi za teren) i onda za svaki izlazni piksel koji senčimo, čitamo susedne piksele i na osnovu njihovih relativnih dubina (Z) izračunavamo koliko nam svetlosti oni blokiraju. To se tipično implementira kao kratak ray tracing/marching kroz to polje dubina (Z-buffer). Pošto je čak i ovaj proces dosta računski i memorijski intenzivan, on se često obavlja na manjoj rezoluciji od rezolucije ekrana i sa vrlo ograničenom preciznošću i broju uzoraka, gde se onda rezultat bluruje i skalira nazad na punu veličinu ekrana. Krajnji cilj je da se ova vrednost upotrebi umesto konstantne boje ambijenta, kako se tradicionalno radilo u real-time 3D grafici.

Chapter 8: Finding Next Gen – CryEngine 2

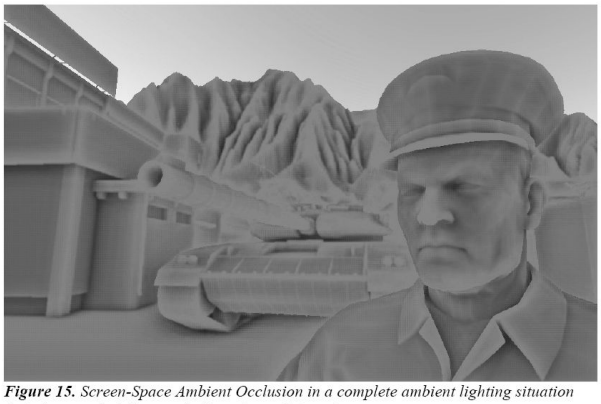


Figure 15. *Screen-Space Ambient Occlusion in a complete ambient lighting situation (note how occluded areas darken at any distance)*

Korisni linkovi:

- https://john-chapman-graphics.blogspot.com/2013/01/ssao-tutorial.html
- https://learnopengl.com/Advanced-Lighting/SSAO

Ostale teme

Mnoge druge teme dolaze u obzir kao specijalne teme po dogovoru

Bodovanje

Bodovanje nije strogo podeljeno i domaći se boduje kao celina. Domaći nosi 40 bodova

Predaja domaćeg

Domaći je potrebno predati barem dva dana pre odbrane slanjem projekta na mail mveniger@raf.rs. Drugi domaći mora biti odbranjen pre početka januarskog ispitnog roka
Pravila za slanje:

- Subject maila treba da bude RG19D2 ime prezime indeks (na primer RG19D2 Marko Veniger RN 5 2014)
- Sadržaj maila treba da bude samo zipp-ovan projekat nazvan isto kao i subject + .zip
- Obavezno je direktorijum u kojem se nalazi projekat nazvati po istom paternu (najlakse bi bilo preimenovati RAFGL direktorijum sa vežbi i onda desni klik, pa send to Compressed)
- Pre slanja obrisati sve izvršne fajlove (ovo nije obavezno zbog pravila, već gmail vam neće dozvoliti da šaljete atačment ako sadrži ovakve fajlove)

Za svako prekršeno pravilo slanjase dobija (kumulativno) 3 negativna boda. Domaći minimalno nosi 0 bodova a maksimalni 40.

Odbrana domaćeg

Odbrana domaćeg je obavezna i osvojeni bodovi se određuju na odbrani. Za odbrane će biti rezervisani termini na fakultetu i radiće se po rasporedu koji će biti objavljen pred kraj semestra. U slučaju da ne možete doći na termin odbrane potrebno je to javiti pre formiranja rasporeda (tačan datum i vreme će biti javljeni)

Odbrana se može raditi na fakutetskim računarima ili na računarima koje ponesete sa sobom.
Odbrana će trajati od 10 do 15 minuta po studentu i obuhvataće kratko demonstriranje rada domaćeg, kratko demonstriranje i objašnjavanje koda domaćeg, potencijalna pitanja vezana za kod ili tehnike primenjene u domaćem.

U slučaju prepisivanja od drugih studenata (otkriveno uz pomoć identičnih ili skoro identičnih kodova, pre odbrane) svi uključeni studenti padaju ispit (nemaju dovoljno predispitnih bodova da izađu na ispit)

U slučaju prepisivanja otkrivenog na odbrani (loše poznavanje implementacije i korišćenih tehnika) student ima pravo na modifikaciju projekta u realnom vremenu (slično kao kolokvijum, ali zbog specifičnosti svakog domaćeg, zadatak modifikacije se zadaje na osnovu projekta studenta u pitanju). Ako student ne uspe da uradi modifikaciju pada ispit (bez drugog domaćeg ne može imati uslov za izlazak na ispit)

Odbrana nema krajnji rok ali je preporučeno da se završi što je pre moguće.
Odbrana se radi sa asistentom uz moguće prisustvovanje profesora.

Sve dodatne informacije možete saznati putem maila (mveniger@raf.rs) ili na konsultacijama.