Grafikkprogrammering – lysmodeller

Denne uken skal du implementere lys og sammenlikne to lysmodeller.

Normalvektorer

For å håndtere lys og shading trenger vi normalvektorer for hver verteks. Det er flere måter å representere normalvektorer på, men den enkleste (for oss) er per-verteks representasjon. Det vil si at vi antar at 3D objektet er på forhånd lagret med informasjon for normalvektor for hver verteks. Hvis du ikke har dette på plass fra tidligere, så må du nå inkludere støtte for normalvektorer (parsing fra OBJ og sende disse til GPU og verteks-shaderen).

Antatt at du transformerer verteksene med ModelViewProjection, så må også normalvektorene transformeres. Denne transformasjonen er definert ved transpose av den inverterte ModelView-matrisen. Opprett en ny transformasjonsmatrise (du kan kalle den normalsTransform) som er lik $((V * M)^{-1})^T$, og send denne til verteks-shaderen.

Materialegenskaper

Lys samhandler med objektets materialegenskaper og hvordan et objekt ser ut avhenger derfor av dens materialer. For objekt-representasjonen (i JavaScript) av 3D objektet, legg til egenskaper for materialer som inkluderer diffuse, specular, og ambient.

Hvis du vil, kan du inkludere en parser for MTL, filen som følger med OBJ hvis objektet har materialegenskaper og/eller tekstur. I tillegg, burde du opprette UI der bruker kan manipulere objektets materialegenskaper.

PS: I denne oppgaven antar vi at objektet har en type materiale som er konstant over hele objektet. Variasjoner kan imidlertid kodes inn i teksturer hvis du ønsker å utforske dette.

Lys

Diffuse og specular komponentene avhenger av at lys kommer fra en retning, der vi ignorerer distansen mellom lyset og objektet. Denne lysmodellen støtter to typer lys: point lights og directional lights. Velg en av disse (eller begge) og inkluder en representasjon av lys i JavaScript prosjektet. Send dataene for lyset/lysene over til verteks shaderen.

Jeg anbefaler at du animerer lyset slik at lysmodellen visualiseres bedre. Bruker kan for eksempel slå av/på en automatisk animasjon av et lys som roterer rundt objektet (du kan bruke spherical coordinates/kulekoordinater for å oppnå dette). Dette kan imidlertid legges til etter at du har implementert støtte for lyset, slik at du ser endringene dine ☺

Ambient shading

I følgende oppgaver arbeider du mer aktivt med shader-programmering. For å lære dette bedre, ikke hent løsninger fra Internett! All informasjon du trenger blir gitt i forelesning. Sjekk referanse for WebGL/GLSL hvis det trengs:

http://www.cheat-sheets.org/saved-copy/webgl-reference-card-1_0.pdf

Start å implementere lysmodellen med enkel ambient shading. Lysstyrken for bakgrunnslyset burde komme utenfra verteks-shaderen, som for eksempel fra en UI slider i webapplikasjonen.

Diffuse shading (Lambertian)

Implementer komponenten for diffuse (Lambertian) shading. Alle kalkuleringer skal gjøres i verteks-shaderen, dvs. at fargen som kalkuleres for verteksen sendes til fragment-shaderen som er en enkel pass-through shader.

Husk at vektorene i dot-produktet må være normaliserte.

Ta hensyn til dot(L,N) < 0 (lyset er bak objektet). Du kan bruke max funksjonen for dette (max(0, <>))

Specular shading

Implementer komponenten for specular shading. GLSL støtter en funksjon *reflect*, som kan brukes til å kalkulere R-vektoren. Denne modellen krever en retningsvektor fra kameraet ditt. Påse igjen at vektorene i dot-produkt kalkuleringen er normaliserte.

Etter at du har implementert lysmodellen, gjør eventuell refaktorering i verteks-shaderen. For eksempel, kan shaderen inneholde mange variabler som enten kan innkapsles i objekter (structs i GLSL) eller arrays. Du kan også legge til støtte for flere lys nå.

Per-vertex vs. Per-pixel shading

Lag en ny shader som gjennomfører per-pixel shading (Phong shading). Behold din forrige per-vertex modell slik at du kan enkelt sammenlikne mellom disse (bruker burde kunne dynamisk bytte mellom de to lysmodellene).

Per-pixel shading involverer at du i vertex-shaderen kalkulerer eventuelle vektorer du trenger i fragment shaderen (som normalvektoren). Disse vektorene sendes fra vertex til fragment shader. Selve kalkuleringen (dot-produkt etc.) gjennomføres dermed i fragment shaderen.

Denne oppgaven krever at du forstår forskjellene mellom vertex og fragment shader og ut ifra dette hva som må kalkuleres i vertex-shader og hva som *kan* kalkuleres i fragment-shader.