UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Eksamen i	INF3320/INF4320 — Metoder i grafisk databehandling og diskret geometri
Eksamensdag:	7. desember 2006
Tid for eksamen:	15:30 – 18:30
Oppgavesettet er på 4 sider.	
Vedlegg:	Ingen
Tillatte hjelpemidler:	Ingen

Kontroller at oppgavesettet er komplett før
du begynner å besvare spørsmålene.

Oppgave 1 Transformasjoner

Vi har et kamera i $[-1, 0, 0]$ som ser i retningen $[-1, 0, 0]$ med $[0, 1, 0]$ som "opp-retning". I tillegg har vi et bounding-volum i form av en kule, med senter i $[-5, 0, 0]$ og med radius 2. Vi starter med `MODELVIEW` og `PROJECTION`-matrisene satt til identitetsmatriser. Vi skal sette opp disse to matrisene slik at kameraet er slik som beskrevet. I tillegg skal frustumet *være så lite som mulig*, men inneholde hele bounding-volumet. Vinduet er kvadratisk.

Skriv ned `OpenGL`-kode som setter opp `MODELVIEW` og `PROJECTION`-matrisene, samt eventuell mellomregning som trengs for å regne ut parametere. Hint: $\sin(30°) = \frac{1}{2}$, $\cos(30°) = \frac{1}{2}\sqrt{3}$, og $\tan(30°) = \frac{1}{3}\sqrt{3}$.

Løsning *Modelview-matrisen bygges med*

```
glMatrixMode(GL_MODELVIEW);
glRotate*( -90.0, 0.0, 1.0, 0.0 );
glTranslate*( 1.0, 0.0, 0.0 );
```

Øyepunktet er 4 *fra origo av kulen. Siden radius er* 2*, så plasserer vi nærplanet ved* 2 *og fjernplanet ved* 6*. Videre, sidekantene av frustumet skal tangere med denne kulen, så finner vi vinkelen utifra senteraksen til sidekantene* α *med*

$$\sin(\alpha) = \frac{2}{4}, \qquad \alpha = 30°$$

Da kan vi finne lengden ut fra senteraksen ved nærplanet h *med*

$$h = 2\tan(30°) = \frac{2}{3}\sqrt{3}.$$

Frustumet blir da bygget med

(Fortsettes på side 2.)

```
glMatrixMode(GL_PROJECTION);
glFrustum( -h, h, -h, h, 2.0, 6.0);
```

Oppgave 2 Depth-buffer

I forrige oppgave ble du bedt om å spesifisere et frustum som var så lite som mulig. Hvorfor er det så viktig å være nøye med å sette nær og fjern-planene når man bruker depth-buffer? Hva galt kan skje?

Løsning *Depth-bufferet har en gitt presisjon, og hvis to verdier blir kvantisert til samme verdi kan man ikke si hvilke av de to verdiene som er nærmest kameraet. I tillegg, siden det er vanligvis* $\frac{1}{z}$ *som blir lagret i depth-bufferet, så er det mye mer presisjon når* z *er liten enn når* z *er stor. Følgelig er det viktig å flytte nærplanet så langt unna kameraet som mulig for å få utnyttet presisjonen. Med for lite presisjon i depth-bufferet, får man effekt som kalles "z-fighting". Dette manifesterer seg som at man får pikseler fra bakenforliggende geometri som kommer på forsiden av forenforliggende geometri.*

Oppgave 3 Lysmodeller

Vi skal se på halvvektorformuleringen av Phongs lysmodell. Tegn en skisse av de involverte vektorene og skriv ned de relevante uttrykkene for å lyssette et punkt med denne lysmodellen.

Løsning *Halv-vektoren finner vi med* $\mathbf{h} = (\mathbf{l} + \mathbf{v})/\|\mathbf{l} + \mathbf{v}\|$ *hvor* $\mathbf{v}$ *er synsvektoren. Den resulterende farven blir da*

$$\mathbf{c}_{out} = \mathbf{c}_{amb} + \max(\mathbf{n} \cdot \mathbf{l}, 0.0)\mathbf{c}_{diff} + \exp(\max(\mathbf{n} \cdot \mathbf{h}, 0.0), s)\mathbf{c}_{spec},$$

hvor $\mathbf{c}_{diff}$ *er resultatet av komponentvis multiplikasjon av materialets diffus-farve og lyskildens diffusefarve, etc., og* s *er materialets "shininess".*

Oppgave 4 Bezierkurver

La $\mathbf{p} : [0, 1] \to \mathbb{R}^2$ og $\mathbf{q} : [0, 1] \to \mathbb{R}^2$ være to Bezierkurver

$$\mathbf{p}(t) = (1-t)^3\mathbf{p}_0 + 3t(1-t)^2\mathbf{p}_1 + 3t^2(1-t)\mathbf{p}_2 + t^3\mathbf{p}_3,$$
$$\mathbf{q}(t) = (1-t)^3\mathbf{q}_0 + 3t(1-t)^2\mathbf{q}_1 + 3t^2(1-t)\mathbf{q}_2 + t^3\mathbf{q}_3,$$

med kontrollpunkter $\mathbf{p}_0 = (-1, 0)$ $\mathbf{p}_1 = (-1, 1)$, $\mathbf{p}_2 = (0, 1)$, $\mathbf{p}_3 = (0, 0)$, og $\mathbf{q}_0 = (0, 0)$ $\mathbf{q}_1 = (0, -1)$, $\mathbf{q}_2 = (1, -1)$, $\mathbf{q}_3 = (1, 0)$. La $\mathbf{r} : [0, 2] \to \mathbb{R}^2$ være splinekurven definert ved

$$\mathbf{r}(t) = \begin{cases} \mathbf{p}(t) & 0 \le t \le 1, \\ \mathbf{q}(t-1) & 1 < t \le 2. \end{cases}$$

(Fortsettes på side 3.)

Skisser $\mathbf{r}$. Hvilken kontinuitetsorden har $\mathbf{r}$, dvs. hva er den største k slik at $\mathbf{r}$ er C^k for $t = 1$?

Løsning *$\mathbf{r}$ er minst C^0 fordi $\mathbf{p}_3 = \mathbf{q}_0$. $\mathbf{r}$ er minst C^1 fordi $\mathbf{p}_3 - \mathbf{p}_2 = (0, -1) = \mathbf{q}_1 - \mathbf{q}_0$. Men $\mathbf{r}$ er ikke C^2 fordi C^2 krever at $\mathbf{p}_1 - 2\mathbf{p}_2 + \mathbf{p}_3 = \mathbf{q}_0 - 2\mathbf{q}_1 + \mathbf{q}_2$ og*

$$\mathbf{p}_1 - 2\mathbf{p}_2 + \mathbf{p}_3 = (-1, -1), \qquad \mathbf{q}_0 - 2\mathbf{q}_1 + \mathbf{q}_2 = (1, 1).$$

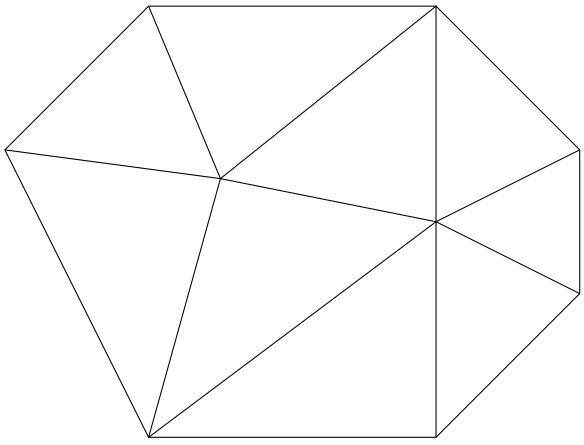
Oppgave 5 Triangulering av polygoner

Beskriv en metode for å triangulere et vilkårlig polygon P i planet. P har en enkel rand (uten selvskjæringer) men er ikke nødvendigvis konveks.

Løsning *En metode er som følger. Hvis P har tre noder er det en entydig triangulering som består av trekanten dannet fra de tre nodene. Ellers finner vi et "øre", dvs. en trekant $E = [\mathbf{v}_{i-1}, \mathbf{v}_i, \mathbf{v}_{i+1}]$ slik at den indre vinkelen i $\mathbf{v}_i$ er mindre enn π (mindre enn 180 grader) og slik at E er "tom", dvs. det finnes ingen andre noder $\mathbf{v}_j$ som ligger inni E. Det er et teorem ('The two ears theorem') som fastslår at det alltid finnes minst et øre (faktisk minst to). Vi danner et nytt polygon P' ved å fjerne $\mathbf{v}_i$ fra P, d.v.s., ved å erstatte de to kantene $[\mathbf{v}_{i-1}, \mathbf{v}_i]$ og $[\mathbf{v}_i, \mathbf{v}_{i+1}]$ med $[\mathbf{v}_{i-1}, \mathbf{v}_{i+1}]$. Vi triangulerer P' (rekursivt) og til slutt legger E til trianguleringen.*

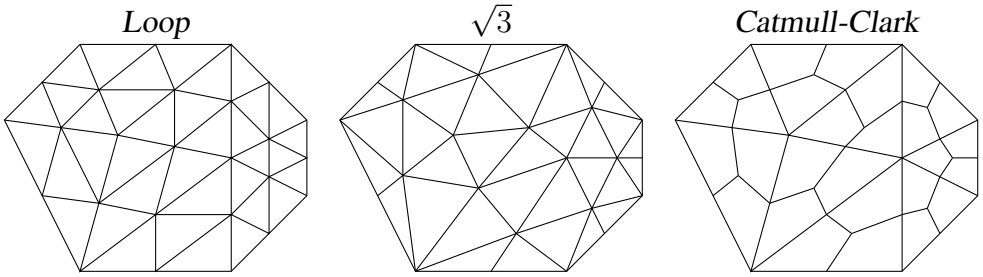
Oppgave 6 Subdivisjonsflater

Lag tre skisser av konnektiviteten til meshet nedenfor efter en subdivisjon med henholdsvis Loop-, $\sqrt{3}-$ og Catmull-Clark-skjemaene.



(Fortsettes på side 4.)

Løsning



Oppgave 7 Arkitektur

Anta OpenGL 2.0 uten utvidelser. Skisser hovedkomponentene i den logiske pipelinen. Skriv ned hvilke av disse stegene som er programmerbare. I tillegg, skriv ned hva slags type data som går inn og ut av hvert steg, og hva slags data man kan spesifisere for de programmerbare stegene.

Løsning *Vi har følgende steg:*

1. *Vertex transformasjoner (programmerbar). Vertex-attributer inn, transformerte vertexer ut.*
2. *Primitive assembly. Begin/end-state inn, transformerte vertexer inn, primitiver ut.*
3. *Klipping. Primitiver inn, primitiver ut.*
4. *Rasterisering. Primitiver inn, fragmenter ut.*
5. *Fragmentoperasjoner (programmerbar). Fragmenter inn, fragmenter ut.*
6. *Buffer-operasjoner. Fragmenter inn, oppdaterer framebufferet.*

I OpenGL 2.0 så er Vertex-transformasjoner og fragmentoperasjoner programmerbare. Vertex-transformasjoner får vertex-attributter inn og gir transformerte vertex'er (med eventuelle attributter) ut. Disse attributtene blir kalt "varying" fordi de blir interpolert over primitiver. Fragmentoperasjoner får disse interpolerte varying-attributtene inn og gir den endelige farven og z-posisjonen til fragmentet ut.

For å spesifisere konstanter, så bruker man enten teksturer eller "uniform"-variable.