Examen Final de VIG — 2009-10, Q2
17/06/2010, 11:30

Disposeu de dues hores.

Contesteu les preguntes en l'espai proporcionat en aquest enunciat. Si és estrictament necessari, afegiu un full pel que no us hagi cabut; en aquest cas, la continuació de cada pregunta s'ha de fer en un full diferent. Tingueu en compte, tanmateix, que **valorarem la concisió** i brevetat a més de —naturalment— la completesa i l'exactitud.

Tanmateix, òbviament, **cal que justifiqueu totes les respostes** que doneu. Una resposta correcta però sense justificació o amb una justificació incorrecta no dóna punts.

Per a contestar aquest examen no podeu consultar cap material addicional.

Cognoms: _____ Noms: _____

Pregunta:	1	2	3	4	5	6	7	8	9	Total
Punts:	10	5	20	10	10	10	10	5	20	100
Obtinguts:										

1. Tenim una paret il·luminada per un únic focus de llum. La il·luminació es calcula usant únicament el model d'il·luminació de Phong. La paret està pintada amb una pintura brillant. Si s'està dibuixant amb OpenGL amb colorat `GL_SMOOTH`, veurem diferències si la geometria de la paret és un únic polígon rectangular o si està formada per una malla densa de triangles? Dóna una explicació detallada de perquè. [10]

   Solució: El model d'il·luminació de Phong simula l'efecte d'especularitat en materials brillants, és a dir les petites taques brillants que corresponen a la reflexió (borrosa) de fonts de llum. Però el colorat `GL_SMOOTH` sols l'aplica als vèrtexs dels polígons, i calcula el color de cada punt del polígon per interpolació. Per tant, si pintem la paret com un sol polígon, les taques especulars al centre de la paret es perdran, i si n'hi ha una a un vèrtex, resultarà molt més grossa que el que correspondria. Fent servir una densa malla de triangles, es prenen moltes més mostres sobre la cara de la paret, i per tant el color de cada punt es calcula a partir de mostres molt més properes, facilitant que es puguin veure taques especulars al mig de la paret, i a més es limitarà el creixement artificial de les taques especulars per efecte de la interpolació.

2. Què és i per a què serveix un "varying"en els vèrtex i fragment shaders? [5]

   Solució: `varying` és un atribut d'un tipus que indica que la variable declarada és una variable de sortida (en el cas d'un vertex shader), o d'entrada (en el cas d'un fragment shader). Els valors escrits a un varying per un vertex shader s'interpolen per cadascun dels fragments relacionats amb aquell vèrtex en la rasterització, de la mateixa manera que s'interpolen els colors en el cas del colorat `GL_SMOOTH`.

3. Un professor demana a dos estudiants que li escriguin el codi OpenGL necessari per a definir una càmera que mostra una certa escena en una vista en planta. Els estudiants responen amb els dos codis següents respectivament: [20]

   Codi 1:

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-100, 100, -100, 100, 10, 150);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity ();
gluLookAt(0, 80, 0, 0, 50, 0, 1, 0, 0);
```

   Codi 2:

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-100, 100, -100, 100, 10, 150);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(0, 0, -80);
glRotatef(90, 0, 0, 1);
glRotatef(90, 1, 0, 0);
glRotatef(-90, 0, 1, 0);
```

Es demana:

(a) Compleixen l'objectiu demanat cadascun d'aquests dos codis?

Solució: Sí, tots dos codis implementen una vista en planta, perquè totes dues càmeres estan situades sobre l'eix de les Y (concretament a la posició $(0, 80, 0)$) i mirant de forma perpendicular al pla XZ.

(b) Defineixen tots dos la mateixa càmera?

Solució: Encara que les dues càmeres es troben a la mateixa posició, mirant en la mateixa direcció i que es defineix en tots dos casos el mateix volum de visió, en realitat no són la mateixa càmera perquè difereixen en la seva verticalitat, en el seu vector up. En el codi 1 el vector up és el $(1, 0, 0)$ i per tant l'orientació dels eixos de l'observador serà: X_{obs} en direcció Z_a, Y_{obs} en direcció X_a i Z_{obs} en direcció Y_a. I en el codi 2, el vector up és el $(0, 0, -1)$ i per tant tindrem: X_{obs} en direcció X_a, Y_{obs} en direcció $-Z_a$ i Z_{obs} en direcció Y_a.

(c) Creus que les transformacions geomètriques del Codi 2 es poden optimitzar?

Solució: Sí que es poden optimitzar les transformacions geomètriques del codi 2, perquè per a deixar el sistema de coordenades d'observador orientat tal que $X_{obs} = X_a$, $Y_{obs} = -Z_a$ i $Z_{obs} = Y_a$, hi ha prou amb fer una única rotació de -90 graus sobre l'eix X. Per tant el codi a partir de la línia 6 quedaria:

```
6 glTranslatef(0, 0, -80);
7 glRotatef(90, 1, 0, 0);
```

4. Tenim el color definit per les coordenades HSB (180,0,0.6). Quina seria la seva representació en RGB i en CMY? I si augmentem la seva saturació a 1, com canviaria la seva representació en RGB? 10

Solució: El color representat en HSB com $(180, 0, 0.6)$, ens indica que té un tint (hue) de color cian, una saturació de 0 i una brillantor de 0.6. Com que la saturació és 0, el color només pot ser un gris, i serà un gris amb 0.6 d'intensitat, així que la seva representació en RGB serà $(0.6, 0.6, 0.6)$. En CMY, que és el complementari del RGB seria $(0.4, 0.4, 0.4)$.

Si augmentem la saturació a 1, passem a tenir un color pur, i com que el tint (hue) és de cian, amb intensitat 0.6, la seva representació en RGB serà $(0, 0.6, 0.6)$.

5. Usant la càmera definida en el Codi 1 de l'exercici 3, pintem un triangle de vèrtexs: $V_1 = (100, 70, 0)$, $V_2 = (0, -70, -50)$ i $V_3 = (-100, -70, 100)$. 10

- Dóna els vèrtex en SCO, SCN i SCD, tenint en compte que es té una vista de 800x600.
- Pinta el que es veuria a la vista. Discuteix la relació entre la forma real del triangle i la que es veu a la vista.

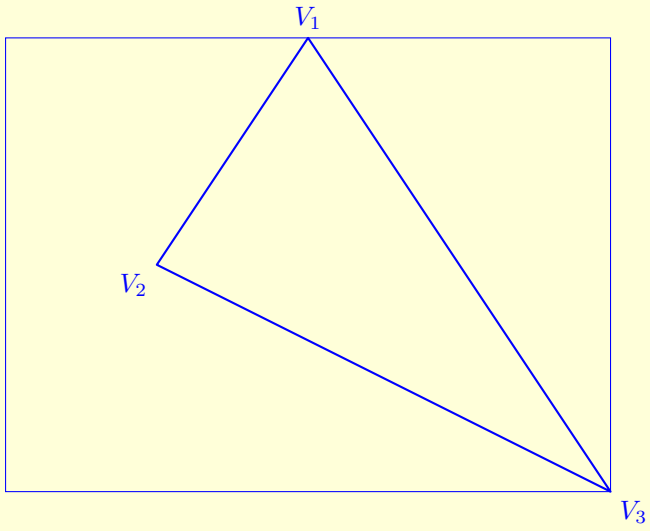
Solució: Donat que, com diu la solució de l'exercici 3, el sistema de coordenades de la càmera es troba centrat al punt $(0, 80, 0)$ i té els eixos orientats de manera que $X_{obs} = Z_a$, $Y_{obs} = X_a$ i $Z_{obs} = Y_a$, les coordenades dels vèrtexs en el sistema de coordenades d'observador seran: $V_{1,o} = (0, 100, -10)$, $V_{2,o} = (-50, 0, -150)$ i $V_{3,o} = (100, -100, -150)$.

Com que el volum de visió és un paral·lelepípede que defineix un window que va des de -100 a 100 en les dues components X i Y i en Z el Z_{near} està sobre la $Z_{\text{obs}} = -10$ i el Z_{far} està sobre la $Z_{\text{obs}} = -150$, les coordenades dels vèrtexs en el sistema de coordenades normalitzat (on $Z_{\text{near},n} = -1$ i $Z_{\text{far},n} = 1$) són: $V_{1,n} = (0, 1, -1)$, $V_{2,n} = (-0.5, 0, 1)$ i $V_{3,n} = (1, -1, 1)$.

I com que la vista va del píxel $(0, 0)$ al píxel $(800, 600)$, les coordenades en sistema de coordenades de dispositiu són: $V_{1,d} = (400, 600, 0)$, $V_{2,d} = (200, 300, 2^{\text{nb}} - 1)$ i $V_{3,d} = (800, 0, 2^{\text{nb}} - 1)$, on nb és el nombre de bits que usa el Z-buffer.

$\Longrightarrow$

El que es veuria a la vista es mostra a la figura:



I pel que fa a la relació entre la forma real del triangle i la que es veu a la vista, el triangle es veu deformat per dos motius, el primer és que els tres vèrtexs de l triangle no es troben en el mateix pla perpendicular a la direcció de visió (no tenen la mateixa profunditat) i per tant la projecció dels vèrtexs fa que les distàncies dels costats es vegin alterades. El sergon motiu pel qual es deforma el triangle també és la diferència entre la relació d'aspecte del window i la de la vista; com que el window té $ra = 1$ i la vista té $ra = 4/3$ el triangle es veurà més estirat en amplada.

6. Tenim una càmera ortogonal definida de la següent manera: 10

```
gluOrtho(-1.0, 1.0, -1.0, 1.0, 1.0, 10.0)
```

Asumint la mateixa posició de l'observador, quin és el mínim angle d'obertura vertical amb el qual s'ha de definir la següent càmera perspectiva, per tal que el volum de visió de la càmera ortogonal estigui totalment inclòs al seu volum de visió?

```
gluPerspective(fovy,1.0, 1.0, 10.0)
```

Solució: Per a què el volum de visió de la càmera ortogonal estigui completament inclòs en el de la càmera perspectiva, el que cal és que comparteixin el window en el pla del Z_{near}. Aquest Z_{near} es troba a distància 1 de l'observador, i la meitat de l'alçada del window de la càmera ortogonal també val 1, així que l'angle que fa que el window de la càmera perspectiva també sigui, sobre el Z_{near} de 2×2 serà el que compleix: $\texttt{fovy} = 2tan^{-1}(\frac{1}{1}) = 90$, i aquest és l'angle mínim necessari que es demana.

7. Donat el tetraedre definit pels vèrtexs: V1=$(0,0,0)$, V2=$(10,0,0)$, V3=$(0,10,0)$ i V4=$(0,0,10)$, i pintant amb omplert de polígons, volem veure a la vista una imatge que ens mostra tres triangles iguals compartint un vèrtex que queda al mig de la vista. Defineix **tots** els paràmetres d'una càmera que ens permeti veure justament aquesta imatge a la vista, i escriu el tros de codi OpenGL que defineixi aquesta càmera. 10

Solució: Aquest tetraedre té tres cares iguals (els tres triangles isòsceles continguts als tres plans coordenats) i una diferent a les demés (el triangle equilàter de vèrtexs V2, V3 i V4). Per tant és raonable triar el vèrtex V1 com a vèrtex a veure al mig de la vista. Plantejarem la solució amb una càmera ortogonal (axonomètrica), per tant no és indispensable triar V1 com a VRP, però resulta més senzill triar un frustum simètric, i posar

$$\text{VRP} = V1 = (0,0,0).$$

$\Longrightarrow$

Per tal que els tres triangles iguals es deformin en igual mesura, cal que la direcció de projecció formi el mateix angle amb cadascun dels eixos coordinats, és a dir colineal amb el vector $(-1,-1,-1)$. El signe negatiu vé imposat perquè volem veure el tetraedre des del quadrant on totes les coordenades són negatives. Per a una càmera axonomètrica, a banda de determinar la direcció correctament, és del tot irrellevant a quina distància posem l'observador sempre que adaptem correctament els plans de retallat anteroposterior. Podem triar, doncs

$$\text{OBS} = (-1,-1,-1).$$

Finalment, per acabar de determinar la posició de la càmera, cal triar un vector VUP. Com l'enunciat no indica com han d'aparèixer els triangles a la pantalla, qualsevol vector que no sigui colineal amb el vector que va de l'observador al VRP serveix. Per exemple, el més popular dels VUPs:

$$\text{VUP} = (0,1,0).$$

Quant al tipus de càmera, ja hem indicat que fariem servir una càmera axonomètrica. Òbviament haurem de fer servir un $Z_n < sqrt(3)$, ja que aquesta és la distància de l'observador al *view reference point*. Al seu torn, la recta que passa per l'observador i per V1 talla la cara oposada a V1 al punt $(10/3, 10/3, 10/3)$, per tant necessitem un $Z_f > sqrt(3) + \frac{10}{sqrt(3)} = \frac{13}{sqrt(3)}$. Si observem que les arestes de la cara V2-V3-V4 són paral·leles al *window*, i mesuren $10sqrt(2)$, resulta natural (suposant una relació d'aspecte igual a 1 al viewport) triar left $= -5sqrt(2) \approx -7.1$, right $= 5sqrt(2) \approx 7.1$, bottom $= -5sqrt(2) \approx -7.1$ i top $= 5sqrt(2) \approx 7.1$.

El codi necessari per a definir aquesta càmera en OpenGL és:

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(-1, -1, -1,    0, 0, 0,    0, 1, 0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-7.1, 7.1, -7.1, 7.1, 1.7, 7.6);
```

8. En un programa, l'autor ha definit, per error, dues llums idèntiques (mateix tipus, posició i colors). En visualitzar una escena amb el programa, tal que les llums la il·luminen correctament, hi haurà alguna diferència apreciable si està activada sols una, o si ho estan les dues? La teva resposta és independent dels paràmetres de color de les llums i de l'escena que visualitzem? [5]

Solució: Anomenem **A** i **B** als dos llums idèntics de l'enunciat. OpenGL calcula el color de cada vèrtex sumant les contribucions de cada llum activa de l'escena. Per tant, si la contribució neta a un vèrtex de **A** no es nul·la, aleshores activar les dues duplicarà aquesta contribució. El resultat es truncarà a 1 si l'excedeix. Per tant, podem assegurar que s'hi veuran diferències entre la imatge amb sols una de les dues llums activada, i amb les dues, sempre que existeixi un vèrtex a l'escena tal que:

- Algun dels polígons que el tenen per vèrtex és visible a la vista
- el color calculat amb totes les llums de l'escena excepte la **B** no és igual a 1 en les tres components de color del vèrtex i
- per una component tal que el color del punt anterior no és >= 1, **A** té alguna emissió en aquesta component diferent de zero, i el material del polígon visible té la corresponent constant de color també diferent de zero.

9. Volem utilitzar l'algorisme de selecció d'OpenGL en una escena on hi ha, com a molt, 100 objectes, emmagatzemats al vector `Vobjectes`. Cada objecte está identificat amb 1 enter que indica la posició de l'objecte dins del vector. Podeu suposar que tenim les funcions `setModelview()` i `setProjection()` —que donen valors a les matrius corresponents de forma correcta— ja programades. Al codi de l'aplicació que permet navegar interactivament per l'escena tenim la classe [20]

`Objecte` que emmagatzema informació de les cares i els vèrtexs dels objectes i que disposa del mètode `render()` que permet pintar-los.

La selecció la farem amb un rectangle definit sobre la pantalla amb coordenades (retornades al paràmetre del *call-back* corresponent de Qt) $(xsup, ysup)$ i $(xinf, yinf)$. Tots els objectes que caiguin total o parcialment dins d'aquest rectangle quedaran seleccionats.

Heu d'implementar la funció `seleccionaPerArea(int xsup, int ysup, int xinf, int yinf)` que faci la selecció dels objectes. Podeu suposar que teniu un vector `Vobjectes` que conté tots els objectes de l'escena i que cada objecte té un *flag* públic que ha d'indicar si està seleccionat o no. Ens proporciona algun avantatge desactivar el càlcul de la il·luminació per a fer la selecció?

Solució:

```
void seleccionaPerArea(int xsup, int ysup, int xinf, int yinf) {
  GLint v[4];
  glGetIntegerv(GL_VIEWPORT, v);
  GLdouble m[16];
  glGetDoublev(GL_PROJECTION_MATRIX, m);
  // Modifiquem la proj. per a limitar-la al nou volum:
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPickMatrix((xsup+xinf)/2, v[3]-(ysup+yinf)/2,
                xsup-xinf, ysup-yinf, v);
  glMultMatrix(m);
  // Usarem un unsingned per objecte com a nom, pel que els
  // registres dels hits tindran quatre components:
  GLuint selBuff[4*100];
  glSlectBuffer(4*100, selBuff);
  glInitNames(); glPushName(0);
  glRenderMode(GL_SELECT);
  for (int ob=0; ob<Vobjectes.size(); ++ob) {
    glLoadName(ob);
    Vobjectes[ob].render();
    Vobjectes[ob].selected = false;
  }
  glPopName();
  int hits=glRenderMode(GL_RENDER);
  for (int i=0; i<hits; ++i) {
    Vobjectes[selBuff[4*i+3]].selected = true;
  }
}
```