Examen Final de VIG — 2009-10, Q1
15/1/2010, 12:00

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.
Tanmateix, naturalment, **cal que justifiqueu 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	Total
Punts:	10	10	10	10	25	15	10	10	100
Obtinguts:									

1. Suposeu una escena en OpenGL formada per dos cubs de costat 5 amb centres situats a $(5,0,0)$ i $(20,0,0)$, i cares perpendiculars als eixos de coordenades. Tenim una llum *spot* situada al $(0,0,0)$, amb angle d'obertura de 30 graus i amb direcció $(1,0,0)$. Explica raonadament quines cares dels cubs es veuran il·luminades pel focus de llum. [10]

Solució: Si deixem de banda la llum ambient, per a resultar il·luminada una cara, ha de tenir almenys un vèrtex tal que el vector normal de la cara i el vector que uneix aquell vèrtex amb el focus de llum (al $(0,0,0)$) formen un angle de menys de 90 graus. Les úniques cares candidates, per tant, són les cares paral·leles al pla $y-z$ que passen pels punts $(2.5,0,0)$ i $(17.5,0,0)$. Tanmateix, la primera és massa a prop del focus, i donada la seva obertura de sols 30 graus, tots quatre vèrtexs queden fora del seu con de llum, pel que aquesta cara, la més propera a la llum, apareixerà completament fosca. La segona, en canvi, sí que estarà il·luminada (d'un color uniforme), ja que OpenGL fa servir un càlcul local de la il·luminació, que no és capaç de detectar ombres.

2. Explica en què consisteix el *shading* de Gouraud, i en què el *shading* de Phong. Explica quin és el seu propòsit. Quin fa servir OpenGL? [10]

Solució: Els algoritmes de *shading* o de colorat, calculen el color dels diferents fragments resultants de rasteritzar una primitiva.

El de Gouraud, que és usat a OpenGL quan triem el *ShadingModel* `GL_SMOOTH`, calcula aquests colors interpolant linealment els colors dels vèrtexs de la primitiva. Pot fer-se servir tant amb com sense il·luminació. Si es fa servir amb models empírics d'il·luminació, no reprodueix les taques especulars, llevat de si ocorren a algun vèrtex (perquè sols allí s'apliquen aquests models empírics), i a més distorsiona la seva mida.

El de Phong, que no s'ha de confondre amb el model empíric d'il·luminació del mateix nom, calcula el color de cada fragment aplicant les fórmules dels models empírics. A cada fragment es fa servir com normal a la superfície el vector que s'obté d'interpolar les normals a cada vèrtex. Aquest no té perquè ser unitari, i pot donar resultats molt distorsionats si les normals als vèrtexs són molt diferents, i a més el cost de càlcul és molt més elevat. Però té l'avantatge respecte a l'algoritme de Gouraud de poder representar taques especulars àdhuc al bell mig d'una cara.

3. Partint del codi mostrat abaix, i sense fer servir les comandes `glPushMatrix()`/`glPopMatrix()`, indica quin codi inclouries, i a on, per a definir els següents dos focus de llum: [10]

   1. `GL_LIGHT0`, un llum de càmera de color blanc, i
   2. `GL_LIGHT1`, un llum d'escena de color vermell.

   Raona la teva resposta.

```
1  void GLWidget::paintGL( void )
2  {
3    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
4
5    glMatrixMode(GL_MODELVIEW);
6    glLoadIdentity();
7    glTranslatef(0,0,-dist);
8    glRotatef(angleZ,0,0,1);
9    glRotatef(angleX,1,0,0);
10   glRotatef(angleY,0,1,0);
11   glTranslatef(-VRP.x,-VRP.y,-VRP.z);
12
13   // Dibuixa l'escena:
14   ...
15 }
```

Solució: Per a definir un llum de càmera, passant-li la posició en coordenades d'observador, a la matriu *Modelview* hi ha d'haver la identitat en el moment que donem la posició del focus de llum. Per contra, si volem un llum d'escena, a la matriu *Modelview* hi ha d'haver les transformacions (matriu TC) que es fan als vèrtexs de l'escena.

Suposant les posicions següents definides respecte a càmera i escena respectivament:
`GLfloat pos1[4]={0,0,0,1}; GLfloat pos2[4]={0,20,0,1};`

Haurem de posar, entre les línies 6 i 7 el codi: `glLightfv (GL_LIGHT0, GL_POSITION, pos1);`

I a la línia 12 el codi: `glLightfv (GL_LIGHT1, GL_POSITION, pos2);`

Per al color, el codi pot anar en qualsevol lloc abans de pintar l'escena, i de fet, millor que no sigui en el paintGL:

```
GLfloat blanc[4]={1,1,1,1}; GLfloat vermell[4]={1,0,0,1};
glLightfv (GL_LIGHT0, GL_DIFFUSE, blanc);   //només difusa i especular
glLightfv (GL_LIGHT0, GL_SPECULAR, blanc);  //perquè ambient no cal
glLightfv (GL_LIGHT1, GL_DIFFUSE, vermell);
glLightfv (GL_LIGHT1, GL_SPECULAR, vermell);
```

4. Explica què fan cadascuna de les següents primitives d'OpenGL. [10]

   1. `glEnable(GL_COLOR_LOGIC_OP);`
   2. `glLogicOp(GL_XOR);`
   3. `glDrawBuffer(GL_FRONT);`

   Discuteix a més la seva aplicació en un exemple.

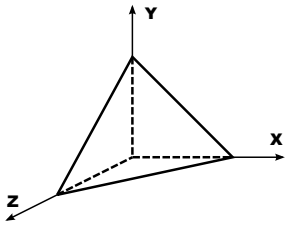
Solució: La primera crida activa el mecanisme pel qual les actualitzacions al *buffer* en què estiguem dibuixant es fan assignant-hi el resultat d'una operació lògica entre el nou valor del fragment que arriba i el ja emmagatzemat al buffer.

La segona crida tria quina ha de ser aquesta operació lògica. Per defecte l'operació és assignar el nou valor, pel que no hi ha diferència entre que `GL_COLOR_LOGIC_OP` estigui activada o no. Amb aquesta crida estem indicant que es faci servir com a nou valor al *frame buffer* el resultat de fer un o exclusiu entre el valor de color del fragment que arriba i el ja guardat al *buffer*. ⟶ ⟶

La tercera instrucció selecciona el *front buffer* com a destí de les operacions de dibuix (a diferència de l'habitual si s'està fent servir *double buffering*, cas en el qual es dibuixa al *back buffer*, i s'intercanvien els buffers quan es vol ensenyar el següent *frame*).

Aquestes crides es fan servir, per exemple, per a implementar el rubber-banding sense haver de repintar tota l'escena a cada moviment del ratolí. Usant-les, podem dibuixar en el front buffer (el que es veu a pantalla) directament, i repintant amb o exclusiu podem esborrar el que acabem d'ensenyar, per a repintar-lo en una nova posició, sense haver de repintar la resta de l'escena, ja que l'o exclusiu és idempotent.

5. Considereu la piràmide de la figura. Els seus vèrtexs es troben a les coordenades $(0,0,0)$, $(10,0,0)$, $(0,10,0)$, $(0,0,10)$. [25]



(a) Implementa una funció `pintaPiramide()` que faci les crides a OpenGL necessàries per a pintar aquesta piràmide. No cal fixar les propietats de material de l'objecte, però cal tenir en compte que la piràmide s'ha de veure correctament il·luminada si la il·luminació està activada.

Solució: Per a que es pinti correctament amb la il·luminació activada caldrà afegir les normals. Les normals de les tres cares perpendiculars als eixos és poden obtenir de manera directa. La quarta cara és perpendicular al vector $(1,1,1)$, que normalitzat queda $(\frac{\sqrt{3}}{3}, \frac{\sqrt{3}}{3}, \frac{\sqrt{3}}{3})$ on $\frac{\sqrt{3}}{3} = 0.577$.

```
void pintaPiramide()
{
   glBegin(GL_TRIANGLES);
   // Cara perpendicular a X
   glNormal3f(-1, 0, 0);
   glVertex3f(0, 0, 0);
   glVertex3f(0, 0, 10);
   glVertex3f(0, 10, 0);
   // Cara perpendicular a Y
   glNormal3f(0, -1, 0);
   glVertex3f(0, 0, 0);
   glVertex3f(10, 0, 0);
   glVertex3f(0, 0, 10);
   // Cara perpendicular a Z
   glNormal3f(0, 0, -1);
   glVertex3f(0, 0, 0);
   glVertex3f(0, 10, 0);
   glVertex3f(10, 0, 0);
   // Cara perpendicular a (1, 1, 1)
   glNormal3f(0.577, 0.577, 0.577);
   glVertex3f(10, 0, 0);
   glVertex3f(0, 10, 0);
   glVertex3f(0, 0, 10);
   glEnd();
}
```

(b) Defineix TOTS els paràmetres d'una càmera ortogonal que quan es pinti la piràmide permeti veure a la vista un triangle equilàter amb la base horitzontal i de manera que es vegi centrat en un window de dimensions 20×16. Escriu el tros de codi OpenGL que defineix aquesta càmera usant transformacions geomètriques per a definir la matriu `GL_MODELVIEW`

Solució: Els paràmetres que caldrà donar per una càmera ortogonal seràn:

- Els límits de la càmara ortogonal: *left*, *right*, *bottom*, *top*, *zNear*, *zFar*.
- Posició de la càmera: *OBS*.
- Target de la càmera: *VRP*.
- Vector cap a amunt: *up*.

Posem que donem als tres punts que formen el triangle més gran de la piràmide els següents noms: $A = (10, 0, 0)$, $B = (0, 10, 0)$ i $C = (0, 0, 10)$. Donat que els límits de la càmera estan en coordenades d'observador i que el window té dimensions 20×16, podem obtenir que $left = -10$, $right = 10$, $bottom = -8$ i $top = 8$. Per a calcular zNear i zFar necessitarem la distància de l'observador al triangle ABC.

Si agafem com a base del triangle ABC l'aresta AC, l'aresta AC haurà de ser paral·lela a l'eix X de l'observador, el segment entre el punt B i el centre de l'aresta AC haurà de ser paral·lel a l'eix Y de l'observador, i el VRP haurà de trobar-se al centre d'aquest segon segment. El punt mig de l'aresta AC està a $(A + C)/2 = (5, 0, 5)$ i el VRP es troba, per tant, a $((5, 0, 5) + (0, 10, 0))/2 = (2.5, 5, 2.5)$. Per a calcular la posició de l'observador caldrà sumar al VRP la normal multiplicada per una certa distància, per exemple, el podríem posar a $(12.5, 15, 12.5)$. Com a vector up podem fer servir $(0, 1, 0)$.

Donat que la distància entre VRP i OBS és de $\sqrt{10^2 + 10^2 + 10^2} = 10\sqrt{3}$, podem prendre valors de zNear i zFar tals que siguin el primer inferior a $10\sqrt{3}$ i el segon superior a $10\sqrt{3}$. Per exemple, podríem fer servir $zNear = 10$ i $zFar = 20$.

Pel codi OpenGL de la matriu de model fent servir només transformacions geomètriques, ens falta saber els angles θ i ψ. Aquests angles es corresponen amb els angles de les coordenades esfèriques del vector $(1, 1, 1)$ normal del triangle ABC (que també és paral·lel a la direcció de visió). Per tant, els angles són $\psi = arctan(1/1) = 45°$ i $\theta = asin(1/\sqrt{3}) = 35.26°$.

Per tant, el codi de la càmera queda:

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(0, 0, -17.32);  // d = 10 * sqrt(3)
glRotatef(35.26, 1, 0, 0);
glRotatef(-45, 0, 1, 0);
glTranslatef(-2.5, -5, -2.5);
```

(c) Quin podria ser un viewport que permeti veure aquest triangle sense deformació? Quina serà la seva relació d'aspecte?

Solució: El viewport hauria de tenir la mateixa relació d'aspecte que el window, $ra = 20/16 = 1.25$. Per exemple, es podria fer servir un viewport de 500×400.

6. Amb la il·luminació apagada i fent servir OpenGL, dibuixem un triangle tal que els seus tres vèrtexs A, B i C cauen en coordenades de dispositiu $(10, 10)$, $(50, 50)$, $(10, 100)$ respectivament, fent servir una càmera ortogonal. Com a resultat de la rasterització, el píxel que es troba a la posició $(10, 55)$ canvia a un color magenta pur $(0.5, 0, 0.5)$. Què ens permet deduir això del color dels tres vèrtexs A, B i C si el mode de pintat és `GL_FLAT`? I si és `GL_SMOOTH`? Raoneu les respostes. 15

Solució: Tal i com estan disposats els punts A, B, C i P = (10, 55), el punt P és precisament al mig de l'aresta AC.

En mode GL_FLAT tota la cara rep el mateix color, per tant, si el punt P canvia a magenta fosc (0.5, 0, 0.5), el primer dels vèrtexs que s'hagi enviat al pipeline d'entre A, B i C ha de ser d'aquest color. Dels altres dos no podem afirmar res, car no intervenen en el càlcul de la coloració de la cara.

Si, en canvi, està activat el mode GL_SMOOTH, el color del punt P serà el resultat de combinar els colors dels vèrtexs del triangle. Com que P està sobre l'aresta AC, el color de B no tindrà influència sobre P. No podem dir res sobre el color de B. De tota manera, els colors de A i C hauran de ser tals que $(Color_A + Color_C)/2 = (0.5, 0, 0.5)$. Per exemple, podria ser que tots dos siguin magenta fosc $Color_A = Color_C = (0.5, 0, 0.5)$, o podria ser que un fos vermell pur i l'altre blau pur $Color_A = (1, 0, 0)$ $Color_C = (0, 0, 1)$.

7. Suposem que mitjançant rubber-banding s'ha aconseguit un rectangle sobre la finestra de la vostra pràctica, amb la cantonada mínima (la superior-esquerra) al píxel $(50, 50)$ i la màxima (inferior-dreta) al píxel $(250, 150)$ (les dades dels píxels són les donades per Qt com posicions del ratolí). Si el que es vol és usar aquest rectangle per a fer una selecció dels 3 trams de carretera que queden més propers a l'observador de tots els que estan dins d'aquest rectangle, respon raonadament a les següents preguntes: 10

   (a) Descriu breument quins són els passos que has de fer per a fer aquesta selecció a partir de les dades d'entrada. No cal que incloguis codi ni que calculis valors.

   Solució: Els passos a fer per a fer aquesta selecció a partir de les dades donades són:

   1. Desabilitar el càlcul d'il·luminació (opcional)
   2. Definir el buffer de selecció amb prou espai per a tots els trams de l'escena, i indicar-li a OpenGL;
   3. Entrar en mode selecció d'OpenGL;
   4. Inicialitzar la pila de noms que usarem per a identificar els trams
   5. Definir el volum de selecció redefinint la matriu de projecció per a què només inclogui el rectangle donat (més detalls a l'apartat b);
   6. Pintar només els trams de la nostra escena, donant noms a cadascun d'ells i guardant-los a la pila de noms (glPushName(ident); pintarTram(); glPopName(););
   7. Tornar al mode render d'OpenGL i capturar el nombre de hits que s'han produït;
   8. Parsejar el buffer de selecció i guardar-nos la informació dels tres trams que tenen la Zmin més petita, que seran els tres que es troben més aprop de l'observador.

   (b) Escriu el tros de codi OpenGL que cal per a definir el volum de selecció necessari per a fer-ho.

   Solució:

```
int vp[4];
glGetIntegerv (GL_VIEWPORT, vp);
float proj[16];
glGetFloatv (GL_PROJECTION_MATRIX, proj);
glMatrixMode (GL_PROJECTION);
glLoadIdentity ();
// cal invertir la coordenada y del centre del rectangle
// perquè Qt té el 0 de la y a dalt i no a baix com OpenGL
gluPickMatrix (150, vp[3]-100, 200, 100, vp);
glMultMatrix (proj);
```

8. Donada una esfera amb constants de material: $K_a = (0.2, 0.2, 0.2)$, $K_d = (0.8, 0, 0.8)$, $K_s = (1, 1, 0)$. Suposant que s'il·lumina únicament per un focus de llum, indica, justificadament, quins haurien de ser els paràmetres de llum del focus (posició i color, sense distingir entre components ambient, difusa i especular) per a poder observar els següents efectes en l'esfera (en cas que l'efecte no sigui possible també has d'indicar-ho). 10

Solució: Si pensem que la component ambient resultat de la il·luminació de l'esfera prové d'una llum ambient que no correspon al focus de llum, obtindriem la solució 1, mentre que si suposem que la component ambient només la pot provocar el focus de llum de l'enunciat obtindríem la solució 2.

En tots els casos, la posició del focus de llum per a poder obtenir tota la silueta de l'esfera sense que li doni el focus de llum (només amb component ambient) és que el focus de llum estigui situat entre la posició de l'observador i el centre de l'esfera, més aprop de l'esfera que l'observador, perquè així es pot veure tros de l'esfera on la llum del focus no arriba.

(a) S'observa la silueta de l'esfera d'un color gris molt fosc i en la resta de l'esfera una gradació de colors magenta.

Solució: Solució 1: focus de llum magenta (pex: (1,0,1)). Sense component verda en el focus de llum, la component difusa queda magenta (degradat) i la component especular, com que s'afegeix a la difusa també serà magenta.

Solució 2: per a tenir un gris fosc en la component ambient, el focus ha de ser blanc. En aquest cas, s'hauria de veure, a més de la gradació de magentes una taca especular de color blanc, per tant l'efecte descrit no és possible.

(b) S'observa la silueta de l'esfera d'un color gris molt fosc i en la resta de l'esfera una gradació de colors vermells amb una taca més lluminosa de color groc.

Solució: Solució 1: focus de llum groc (pex: (1,1,0)). Com que la taca especular s'ha de veure groga i la gradació ha de ser de vermells, la component blava del focus de llum ha de ser 0. Solució 2: per a obtenir una gradació de vermells cal anular la component blava, i no es podria obtenir el gris de la component ambient. L'efecte descrit no és possible.

(c) S'observa la silueta de l'esfera d'un color gris molt fosc i en la resta de l'esfera una gradació de colors magenta amb una taca més lluminosa de color blanc.

Solució: Solució 1: focus de llum blanc (pex: (1,1,1)). Amb un focus blanc, la gradació produïda per la component difusa es veurà magenta, i la taca especular, com que també s'afegeix a la component difusa es veurà blanca perquè afegirà el blau de la component difusa.

Solució 2: focus de llum blanc (pex: (1,1,1)). En aquest cas, com en la solució 1, aconseguim l'efecte amb un focus blanc, i ja ens està bé perquè la component ambient també l'aconseguim amb llum blanca.

(d) S'observa la silueta de l'esfera d'un color gris molt fosc i en la resta de l'esfera una gradació de blaus amb una taca més lluminosa de color cian.

Solució: Solució 1: focus de llum cian (pex: (0,1,1)). Per a aconseguir una gradació de blaus en la component difusa hem d'anular la component vermella, i amb això ja tenim prou per a la component especular perquè a aquesta (que es veuria verda si només fos l'especular) se li afegeix la component difusa, i es produeix la taca especular cian.

Solució 2: Per a obtenir una gradació de blaus hem d'anular la component vermella per complet, i això provocaria que no es pogués obtenir el gris de la component ambient, per tant l'efecte no és possible.