Examen Parcial de VIG — 2009-10, Q1
9/12/2009, 11:30

Disposeu d'1h i 45minuts.

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	15	15	10	10	10	10	20	100
Obtinguts:									

1. Volem afegir una nova funcionalitat a la pràctica per a que permeti que l'usuari pugui girar la càmera en tercera persona respecte de l'eix z de l'observador.

   (a) Indica quins canvis caldrà que facis en el teu `GLWidget` i com queda el codi de definició de la càmera (tant si l'has hagut de modificar com si no). [6]

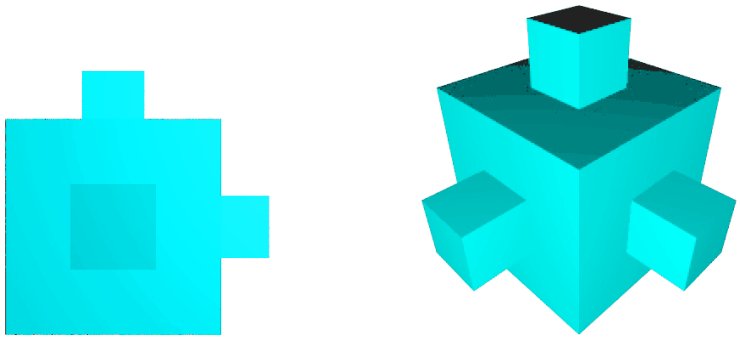
   Solució: Hem d'afegir-hi un atribut `angleZ`, i inicialitzar-ho al constructor a zero. Cal a més inserir un `glRotated(-angleZ, 0., 0., 1.);` entre la crida a `glTranslated(0., 0., -d);` i el primer `glRotate()` de la inicialització de la càmera.
   A més, caldrà afegir un *slot* per a poder actualitzar aquest angle des de la interfície.
   Tot plegat la inicialització de la posició de la càmera serà ara:

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslated(0., 0., -d);
glRotated(-angleZ, 0., 0., 1.);
glRotated( angleX, 1., 0., 0.);
glRotated(-angleY, 0., 1., 0.);
glTranslated(-VRP.x, -VRP.y, -VRP.z);
```

   (b) Quins canvis calen a la interfície per a què s'executi el codi que has afegit a l'apartat anterior? Si et serveix com ajuda pots considerar que tens un *slider* `angleZobs` que té com a rang de valors $[-90, 90]$. [4]

   Solució: Per a aprofitar el `slider` de l'enunciat, caldrà afegir un connect entre el *signal* `valueChanged()` del *slider* i el *slot* que hem afegit al nostre *widget* a l'apartat anterior.

2. Tenim un objecte composat per quatre cubs com es mostra a la imatge: El cub més gros té aresta 3



i es troba situat de tal forma que el seu punt de coordenades mínimes és a l'origen de coordenades. Els tres més petits tenen aresta 1 i es troben sobre les tres cares del primer cub que tenen la normal exterior en la direcció positiva dels tres eixos de coordenades, tal i com es mostra a les imatges.

(a) Suposant que teniu una funció `pinta_cub()` que dibuixa un cub d'aresta 1 centrat a l'origen de coordenades i suposant que la càmera es troba ja inicialitzada, quin codi OpenGL caldria per a dibuixar l'objecte? [10]

Solució:

```
glMatrixMode(GL_MODELVIEW);
glTranslated(1.5, 1.5, 1.5);
glPushMatrix();
glScaled(3., 3., 3.);
pinta_cub();
glPopMatrix();
//
glPushMatrix();
glTranslated(2., 0., 0.);
pinta_cub();
glPopMatrix();
//
glPushMatrix();
glTranslated(0., 2., 0.);
pinta_cub();
glPopMatrix();
//
glPushMatrix();
glTranslated(0., 0., 2.);
pinta_cub();
glPopMatrix();
```

(b) Especifiqueu la seqüéncia de crides que cal fer a OpenGL per a inicialitzar la càmera de manera que s'obtingui una visualització com la de la segona imatge. Feu servir transformacions geomètriques per a especificar la matriu de model (no podeu utilitzar `gluLookAt()`). Podeu suposar que la matriu `GL_PROJECTION` ja ha estat correctament inicialitzada. [5]

Solució:

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslated(0., 0., -7.);
glRotated(30., 1., 0., 0.);
glRotated(-45., 0., 1., 0.);
```

3. Donat el codi OpenGL següent:

```
glViewport (0,0,600,600);
glMatrixMode (GL_PROJECTION);
glLoadIdentity ();
glOrtho (-2,2,-2,2,6,10);
glMatrixMode (GL_MODELVIEW);
glTranslatef (0,0,-10);
```

```
glRotatef (-90,0,0,1);
glRotatef (90,1,0,0);
glTranslatef (-2,0,-2);
glBegin (GL_POLYGON);
glVertex3f (0,4,0);  // Vèrtex V1
glVertex3f (0,4,3);  // Vèrtex V2
glVertex3f (4,4,3);  // Vèrtex V3
glEnd();
```

(a) Quines seran les coordenades dels vèrtexs `V1`, `V2` i `V3` en el Sistema de Coordenades d'Observador (SCO), en Sistema de Coordenades Normalitzat (SCN) i en Sistema de Coordenades de Dispositiu (SCD)? [10]

Solució: Recorrent les transformacions de baix cap a dalt, veiem que els punts seran traslladats primer a les següents coordenades: $V_1^1 = (-2, 4, -2)$, $V_2^1 = (-2, 4, 1)$ i $V_3^1 = (2, 4, 1)$.

A continuació, la rotació de 90 graus al voltant de l'eix x gira la càmera de manera que ara mira en la direcció de les y negatives, amb l'eix y de l'observador en direcció de les z negatives de l'aplicació, i el segon gir al voltant de l'eix z de la càmera (que és la y de l'aplicació) porta l'eix y de l'observador sobre la direcció negativa de l'eix x de l'aplicació.

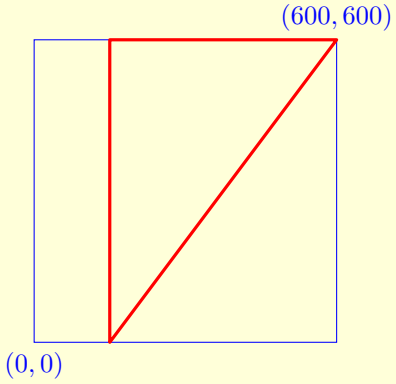
Finalment allunyem l'escena 10 unitats en direcció a la z negativa de l'observador, amb el què ens queda $V_1^{\mathsf{SCO}} = (2, 2, -6)$, $V_2^{\mathsf{SCO}} = (-1, 2, -6)$, $V_3^{\mathsf{SCO}} = (-1, -2, -6)$.

Així, tots els punts es troben al pla de retall anterior, i com la crida a `glOrtho()` defineix el volum de visió com anant des de -2 a 2 tant en x com en y, tenim que $V_1^{\mathsf{SCN}} = (1, 1, -1)$, $V_2^{\mathsf{SCN}} = (-0.5, 1, -1)$, $V_3^{\mathsf{SCN}} = (-0.5, -1, -1)$.

Finalment, com el `viewport` mesura 600×600, i com les fondàries en el SCD són enters de zero al màxim representable, tenim que $V_1^{\mathsf{SCD}} = (600, 600, 0)$, $V_2^{\mathsf{SCD}} = (150, 600, 0)$, $V_3^{\mathsf{SCD}} = (150, 0, 0)$

(b) Dibuixa el que veuries a la vista. [5]

Solució: Aquí mostrem el resultat; en blau veiem la vora del `viewport`, i en vermell el triangle $V_1V_2V_3$:



4. Hem definit amb el computador un dibuix de color cian. Respon raonadament a les següents preguntes (suposa que tots els colors són purs i de màxima lluminositat):

(a) De quin color veurem el dibuix imprès en paper color magenta? [4]

Solució: Com que el paper magenta absorbeix tota la llum verda que hi incideix, i la tinta cian absorbeix tota la llum vermella que hi incideix, el dibuix es veurà de color blau.

(b) Indica quina és la codificació HSB del color que surt quan el dibuix l'imprimim en paper magenta. [2]

Solució: Com el color és blau, l'H serà 240. El color és pur i de màxima lluminositat, per tant S i V seran totes dues iguals a 1.

(c) De quin color veurem el dibuix imprès en paper blanc, il·luminat per una única llum groga? [4]

Solució: La llum groga està feta sols de llum vermella i verda (no conté blau). La tinta cyan absorvirà la component vermella, pel que el dibuix es veurà verd.

5. Escriu el tros de codi necessari per a visualitzar uns cavallets (*tiovivo* en castellà) formats per 6 cotxes situats en cercle al voltant del punt (20,0,20) i en el pla xz ($y = 0$) i formant un cercle de radi R. Disposem del mètode `pinta_cotxe()` que pinta el cotxe amb el centre de la seva capsa mínima contenidora a l'origen de coordenades. La capsa mínima contenidora del cotxe té valors: `xmin`, `xmax`, `ymin`, `ymax`, `zmin` i `zmax`. Suposa que la càmera ja està inicialitzada correctament. [10]

Solució:

```
glMatrixMode(GL_MODELVIEW);
glTranslated(20., 0., 20.);
for (int i=0; i<6; ++i){
   glPushMatrix();
   glRotated(i*60.0, 0., 1., 0.);
   glTranslated(R, (ymax-ymin)/2., 0.);
   pinta_cotxe();
   glPopMatrix();
}
```

6. Sabem que el color blau en HSB es codifica com $(240, 1, 1)$. Si a aquest color li disminuim la intensitat en un 20%, quin color ens quedaria en HSB? Quina seria la seva codificació en RGB? I en CMY? [10]

Solució: El tercer paràmetre del model HSB representa la intensitat, pel que per a reduir-la en un 20% tant sols hem de reduir aquest paràmetre. Per tant en HSB serà $(240, 1, 0.8)$. En RGB, el color original (blau saturat) era el $(0, 0, 1)$, i reduint la seva intensitat en un 20% tenim el $(0, 0, 0.8)$. Per a passar a CMY, finalment, n'hi ha prou amb complementar les components en RGB, i per tant en CMY aquest darrer color és el $(1, 1, 0.2)$.

7. Tenim el següent codi per a implementar el *resize* de la nostra escena. Explica per a cadascun dels casos, quin seria l'efecte que observaríem a la finestra gràfica si fem un *resize*, tant quan la relació d'aspecte (`ra`) sigui major que 1 com quan sigui menor que 1.

(a) [5]

```
ra = 1;
fovy = anglecam;
gluPerspective(fovy, ra, zNear, zFar);
```

Solució: Com que en tots els casos fem servir 1 com a relació d'aspecte en la definició de la càmera, l'usuari veurà una deformació de la imatge sempre que el *viewport* no tingui aquesta mateixa relació d'aspecte. Si en té una de més gran (> 1), la imatge apareixerà estirada horitzontalment, i si és més petita la imatge apareixerà aixafada en el sentit horitzontal (o equivalentment, estirada verticalment).

(b) [5]

```
ra = (float)width/height;
fovy = anglecam;
gluPerspective(fovy, ra, zNear, zFar);
```

Solució: Ara es fa servir correctament la mateixa relació d'aspecte del *viewport*, pel que no hi haurà deformació. Però com no s'ajusta el `fovy`, quan la relació d'aspecte es vagi fent petita es retallarà part de l'escena pels costats.

8. La classe de Qt que implementa el *widget* `QLCDNumber` té, entre d'altres, els següents mètodes:

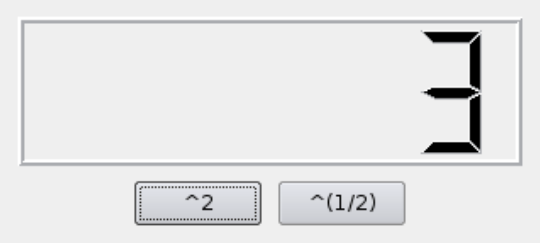
```
public:

  QLCDNumber( QWidget* parent = 0 );
  ~QLCDNumber();
  double value() const;
  int intValue() const;      // Retorna el valor enter mostrat al QLCDNumber
  ...

public slots:

  void display( const QString& s );     // Mostra un string al QLCDNumber
  void display( double num );           // Mostra un real al QLCDNumber
  void display( int num );              // Mostra un enter al QLCDNumber
```

Volem una interfície que tingui un widget amb la mateixa aparença que el `QLCDNumber` i que permeti elevar al quadrat i fer arrels quadrades enteres. Per a això, dissenyarem una interfície amb dos botons (veure la figura), un (que guardarem en una variable anomenada `botoQuadrat`) per calcular



el quadrat del número mostrat, i un altre (que guardarem en una variable anomenada `botoArrel`) per calcular-ne l'arrel.

També farem un *widget* basat en `QLCDNumber` que sigui el marcador, i que guardarem en una variable anomenada `meuComptador`. A aquesta classe l'anomenarem `QLCDCounter`. És a dir, tindríem els següents elements:

```
QPushButton botoQuadrat, botoArrel;
QLCDCounter meuComptador;
```

Inicialment, el marcador ha d'estar inicialitzat a tres. No volem que el nombre visualitzat sigui massa gran i no el puguem representar, per tant, el número mostrat ha d'estar en tot moment dins de l'interval $[0 \ldots 5000]$. En cas d'intentar fer una operació que retorni un valor fora d'aquest rang, el marcador s'inicialitzarà altre cop a tres.

Es demana:

(a) Doneu la definició i implementació de la classe `QLCDCounter`. 15

Solució: Definició (fitxer `QLCDCounter.h`):

```
#include <QLCDNumber>

class QLCDCounter: public QLCDNumber {
  Q_OBJECT

public:
  QLCDCounter(QWidget * parent, const char *name);
  ~QLCDCounter();

public slots:
  void quadrat();
  void arrel();
};
```

Implementació (fitxer `QLCDCounter.cpp`):

```
#include "QLCDCounter.h"
#include <cmath>

QLCDCounter(QWidget * parent, const char *name) : QLCDNumber(parent,name) {
  display(3);
}

~QLCDCounter() {}

void QLCDCounter::quadrat() {
  int num = intValue() * intValue();
  if (num<=5000) display(num);        // no pot ser <0...
  else display(3);
}
void QLCDCounter::arrel() {
  int num = sqrt(intValue());
  if (num<=5000) display(num);        // sqrt() sempre retorna>0.
  else display(3);                    // sols pot ser >5000 si
                                      // un altre component modifica
                                      // també el comptador.
}
```

(b) Especifiqueu en detall les comandes necessàries per a establir les connexions demanades entre els elements de la interfície. 5

Solució:

```
connect( &botoQuadrat, SIGNAL( clicked() ), &meuComptador, SLOT( quadrat() ) );
connect( &botoArrel  , SIGNAL( clicked() ), &meuComptador, SLOT( arrel() ) );
```