Pràctica número 2: Afegir càmera interactiva en el billar

Tema: Implementació del *pipeline* gràfic: a partir de la definició de l'escena, realitzar el control de la càmera i navegació 3D.

Objectiu: Visualització d'una escena 3D amb control de la càmera amb Qt i OpenGL utilitzant *shaders*. Visualització del joc en tercera persona i en primer persona.

La pràctica té com objectiu afegir una càmera dinàmica controlada des de la interfície per a poder obtenir diferents vistes de l'escena de la sala de billar. A més a més s'afegirà una segona càmera en primera persona, per a controlar la direcció de tir de la bola blanca. A la pràctica 1 has construït i has visualitzat el pla base i les boles des d'un punt de vista determinat per GL i les rotacions es feien directament sobre els models. En la pràctica 2, es defineix el concepte **càmera** que permet explorar-los des de diferents punts de vista, amb diferents factors de zoom i variant el centre de projecció (operació de *panning*). Així mateix, es permetrà la visualització en primera persona des del punt de vista de la bola blanca per a poder controlar la direcció de tir en el joc del billar.

L'escena contindrà dues càmeres: **la càmera general**, que permetrà visualitzar el pla base i les boles i **la càmera en primera persona** que controlarà el tir de la bola blanca en el joc del billar. La pràctica 2 es composa de 4 parts:

- Modificació de la classe **Càmera** (proporcionada en el campus virtual de l'assignatura) per incloure el càlcul de les matrius *model-view* i *projection*, que permetin tant la visualització de l'escena completa com parts d'ella, des de la GPU.
- Inclusió de dues càmeres en l'escena: la càmera general i la càmera en primera persona.
- Implementació dels mètodes a la classe **Escena: initCamera, setAnglesCamera, setVRPCamera, setWindowCamera, setDCamera** que permetran la inicialització/modificació de les càmeres i tornar a calcular les matrius *model-view* i *projection* de les càmeres, depenent de les modificacions que es facin en el atributs de la càmera. Utilitzaran mètodes ja implementats de la classe càmera.
- Interacció i activació de les possibles modificacions de les càmeres controlades pel teclat i el ratolí.

El lliurament de la pràctica es realitzarà en el campus virtual i tindrà associat un qüestionari que s'ha de realitzar de forma presencial sobre el codi desenvolupat després del seu lliurament. En el Campus Virtual trobaràs les instruccions de com lliurar una pràctica a l'assignatura.

Aquest enunciat es composa d'una primera part on es detallen cadascuna de les etapes a seguir en el desenvolupament de la programació de la solució. A continuació, es proposen extensions opcionals que es poden implementar per a poder tenir una transició entre càmeres més real.

1. Etapes de desenvolupament de la pràctica

Pas 1. **Baixa codi del campus virtual:** baixa del campus virtual la classe càmera (camera.cpp i camera.h) i afegeix la classe al teu projecte. Baixa també la classe mat.h i substitueix-la per la que tens ara, sempre i quan no hi hagis fet modificacions en la pràctica anterior.

Pas 2. Modificació del *vertex shader*: Cal modificar el vertex shader per a que apliqui les transformacions geomètriques *model_view* i *projection* a cadascun dels vèrtexs. Recordeu que la sortida dels vertex shader ha de donar coordenades homogènies per tot tipus de projeccions, ja siguin paral·leles o perspectives.

Definició i ús de la matriu model-view/projection en el vertex shader
Com fer-ho?   Les matrius de visualització i de projecció són constants per tots el vèrtexs de tots els objectes. Aquests tipus de variables s'anomenen uniform en els *vertex shaders*. A continuació s'explica com definir i usar la matriu *model-view* en el vèrtex shader, però també cal implementar-ho per la matriu projection.

Com es defineix i s'utilitza la matriu model-view en el vertex shader?

Definició:

```
uniform mat4 model_view;
```

Utilització:

```
void main()
 {
  ......
  gl_Position = model_view*vPosition;
  .....
  color = vColor;
  }
```

Pas 3. Modificació de la classe càmera.

La classe càmera conté tota la informació relativa a la càmera. S'ha dividit en diferents tipus que defineixen les diferents característiques de la càmera explicades a teoria. El tipus **VisuSystem** (sistema de visió) permet definir el VRP i els angles de gir. El tipus **PiramProj** (piràmide de projecció) defineix el volum de visió amb el tipus de projecció, la distància a l'observador i els plans anterior i posteriors de *clipping*.

```
typedef enum {PARALLELA = 0, PERSPECTIVA = 1} TipProj;

typedef struct
 {
  double      angy, angx, angz;  /* angles de gir del sistema de coords obser */
  vec4        vrp;               /* view reference point */
  vec4        obs;               /* posicio de l'observador */
  vec4        vup;               /* verticalitat de la càmera */
 } VisuSystem;

typedef struct
 {
  TipProj    proj;              /* tipus de proj: 0 paral.lela, 1 perspectiva */
  double     d;                 /* distancia observador al VRP */
  double     dant, dpost;       /* distancies al pla de retallat anterior  i posterior
des de l'observador*/
  double     alfav, alfah; /* angles d'obertura càmera vertical i horitzontal */
} PiramProj;
```

Els principals atributs de la classe càmera són:

```
    VisuSystem vs;       /* Sistema de visualitzacio  */
    PiramProj piram;     /* Piramide de projeccio     */
    Capsa2D wd;          /* Window                          */
    Capsa2D vp;          /* Viewport                        */

    mat4  modView;        // Matriu model-view de la CPU
    mat4  proj;           // Matriu projection de la CPU
    GLuint  model_view;   // model-view matrix uniform shader variable (GPU)
    GLuint  projection;   // projection matrix uniform shader variable (GPU)
```

És important mantenir la coherència entre aquests atributs. Per exemple, si es canvien els angles que defineixen la posició de l'observador, caldrà canviar la matriu modView en CPU per a que sigui consistent amb aquests canvis.

Dins de la classe càmera trobareu mètodes que permeten el càlcul de diferents paràmetres necessaris en el càlcul de la càmera, per exemple, el que facilita el càlcul de la posició de l'observador a partir del VRP, els angles i la distància de la càmera a l'observador (**CalculObs()**).

Modifica la classe càmera, seguint els següents passos:

3.1. Implementació del pas a la GPU de la matriu model-view/projection des de la classe càmera

Per a poder passar les matrius en el *vertex shader* cal realitzar els següents passos (estan explicats amb la matriu model-view però també caldria fer-los per la matriu de projecció):

Com fer-ho?

Cal implementar els mètodes:

void setModelViewToGPU(QGLShaderProgram *program, mat4 m);
Connecta la matriu model-view (modView) de la CPU amb la matriu de la GPU (model-view) usada en el *vertex shader.*

void setProjectionToGPU(QGLShaderProgram *program, mat4 p);
Connecta la matriu de projecció (proj) de la CPU amb la matriu de la GPU (projection) usada en el *vertex shader.*

void toGPU(QGLShaderProgram *program);
Connecta les matrius de model-view (modView) i de projection (proj) de la CPU amb les matrius de la GPU usades en el *vertex shader.*

Com es defineix i s'utilitza la matriu model-view en el vertex shader?

Si es tenen definides a la CPU les variables:

```
mat4  modView;
GLuint  model_view;
```

I a la GPU, la matriu està definida amb el nom "model_view". El pas de la matriu a la GPU des de la CPU és:

```
model_view = program->uniformLocation("model_view");
glUniformMatrix4fv( model_view, 1, GL_TRUE, modView );
```

3.2. Implementació de diferents utilitats a la classe Camera

Com fer-ho?

Cal implementar els mètodes:

void ini(int ampladaViewport, int alcadaViewport, Capsa3D capsaContenidoraEscena);
Inicialitza els atributs inicials de la càmera, entre els quals és necessari definir el *viewport* i el *vrp*. El mètode `ini(int a, int h, Capsa3D capsaMinima)` rep la mida del viewport actual, que es coneix en la classe glWidget, consultant els atributs `this->size().width()` i `this->size().height()`.

void CalculaMatriuModelView();
Calcula la matriu model-view (modView) a partir dels atributs de la càmera usant les utilitats definides en el fitxer mat.h.

void CalculaMatriuProjection();
Calcula la matriu projection (proj) a partir dels atributs de la càmera usant les utilitats definides en el fitxer mat.h.

void CalculWindow(Capsa3D);
Calcula la Window necessària per a visualitzar tots els objectes continguts en la capsa mínima contenidora passada per paràmetre.

Al final d'aquest pas, encara no podràs provar el codi, ja que cal incloure una càmera a l'escena i modifcar el glWidget que tens actualment.

Pas 4. Modificació de la classe Escena per incloure la càmera general

4.1. Inclusió d'un nou atribut camGeneral a la classe Escena

Com fer-ho?

- Definir un nou atribut anomenat camGeneral de tipus Camera de la classe Escena.
- Modificar la constructora d'escena per a que creï la càmera general, en el moment de construir l'escena.
- Inicialitzar la càmera general per a que estigui al punt 0, 20, 0 de món, miri al punt (0,0,0) de l'escena i verticalitat en Y's.

4.2. Implementació de diferents utilitats referents a la càmera a la classe Escena

Com fer-ho?

Cadascun d'aquests mètodes rebran una càmera (així prepares el codi per a poder tenir més d'una càmera pel punt 6 de l'enunciat).
Els mètodes següents permetran la inicialització de la càmera i la seva modificació segons diferents atributs de càmera. Cal modificar les matrius *model-view* i *projection* de les càmeres, depenent de les modificacions que es facin en el atributs de la càmera. S'utilitzaran mètodes ja implementats de la classe càmera.

```
void iniCamera(bool camGeneral, int ampladaViewport, int alcadaViewport,
               QGLShaderProgram *program);
```

Si el valor del paràmetre és cert, s'inicialitza la càmera general de l'escena. En cas contrari s'inicialitza la càmera en primera persona.

```
void setAnglesCamera(bool camGeneral, float angX, float angY, float angZ);
```

Si el valor del paràmetre és cert, es canvien la posició de la càmera general de l'escena segons els angles d'entrada. Cal actualitzar els atributs que calguin per a deixar coherent tota la informació de la càmera.

```
void setVRPCamera(bool camGeneral, point4 vrp);
```

Si el valor del paràmetre és cert, es canvien el punt on enfoca la càmera general de l'escena segons el punt d'entrada. Cal actualitzar els atributs que calguin per a deixar coherent tota la informació de la càmera.

```
void setWindowCamera(bool camGeneral, bool retallat, Capsa2D window);
```

Si el valor del paràmetre és cert, es canvia la window de la càmera general i tots els atributs depenents d'aquest canvi.

```
void setDCamera(bool camGeneral, float d);
```

Si el valor del paràmetre és cert, es canvia la distància de la càmera general al pla de projecció i tots els atributs depenents d'aquest canvi.

Nota: Si durant la implementació creus que et calen més mètodes, no dubtis en implementar-los i documentar-los convenientment.

Recordeu que si hi han canvis en aquestes matrius, cal tornar-les a passar al vèrtex shader.

Pas 5. Modificació de la classe GlWidget per incloure la càmera general

Des d'aquesta classe es controla la interacció amb l'usuari del widget GL, el moviment del ratolí, el control del teclat, el refresc del pintat, etc.

5.1. Modificació de la classe GlWidget per inicialitzar la càmera general

Com fer-ho?

1. Afegir un atribut cameraActual de tipus booleà, que sigui cert quan s'ha de modificar la càmera general i fals quan s'hagi de modificar la càmera en primera persona. En aquesta primera prova, posa'l a cert.

2. Inclou la creació de la càmera general quan crees una escena

3. En la pràctica 1, s'usava el mètode adaptaObjecteTamanyWidget() per que no es disposava de càmera. A partir de la pràctica 2 no s'utilitzarà més. Seran les càmeres qui controlaran la visualització final. Modifica els mètodes:

```
GLWidget::GLWidget(QWidget *parent): QGLWidget(QGLFormat(QGL::SampleBuffers),
parent)
void initializeGL();
void paintGL();
void resizeGL(int width, int height);
```

4. Fes les crides adients des de la classe *glWidget* per a que s'actualitzi la càmera actual. En aquest punt, després d'haver fet els apartats anteriors, hauries de veure la taula de billar en una vista de planta (TOP).

Nota: Comprova si les matrius arriben bé a la GPU, calculant el valor que realment han de tenir les matrius inicials per a aquesta primera vista.

5.2. Modificació de la classe GlWidget per modificar interactivament la càmera general

Amb el ratolí es desitja controlar el moviment de la càmera, amb les tecles "+" i "-" es té la possibilitat de fer *zoom* i prement la tecla de "Alt" simultàniament amb les tecles de les fletxes es farà *panning* per l'escena.

- arrossegant el ratolí amb el **botó esquerra** pitjat es canvien els **angles** de visió de la càmera, segons el desplaçament. Si hi ha un desplaçament en X de viewport, es canviarà l'angle Y de la càmera i si hi ha un desplaçament en Y de viewport es canviarà l'angle X de la càmera.

- Prement les tecles "+" o "-" es farà un **zoom** de l'escena. Si es prem "+", es farà una ampliació de la visualització amb un factor d'escala fix (per exemple de 0.05). Si es prem "-", es farà una reducció de la visualització amb el mateix factor d'escala.

- prement la tecla de "Alt" simultàniament amb les tecles de les fletxes es farà un **panning 2D**, és a dir, es desplaçarà el centre de projecció segons un factor fixe independentment del desplaçament que hagi fet el ratolí. També es podria implementar el **panning** 3D desplaçant el VRP un increment, però no resulta una interfície molt amigable ni intuïtiva. En aquesta pràctica, implementeu el *panning* 2D.

Com fer-ho?

Per a realitzar aquest apartat, implementa els mètodes de la classe **glWidget**:

```
void setXRotation(int angle);
void setYRotation(int angle);
void Pan(int dx, int dy);
void Zoom (int positiu);
```

Reprograma el mètode `mouseMoveEvent` per permetre la interacció amb el ratolí que es descriu per a canviar els angles de la càmera. Comprova que els moviments de la càmera segueixen les orientacions definides a classe de teoria.

Caldrà que modifiquis els mètodes següents:

```
void mousePressEvent(QMouseEvent *event);
void mouseMoveEvent(QMouseEvent *event);
void keyPressEvent(QKeyEvent *event);
void keyReleaseEvent(QKeyEvent *event);
```

Exemple:

```
void GLWidget::mouseMoveEvent(QMouseEvent *event)
{
    int dx = event->x() - lastPos.x();
    int dy = event->y() - lastPos.y();
    if (event->buttons() & Qt::LeftButton)
    {
        if(lastPos.y()!= event->y() && lastPos.x()!= event->x()) {
            // Moure convenientment la càmera en angle X i/o en angle Y
        } else if(lastPos.y()!= event->y()) {
            // Moure convenientment la càmera en angle X i/o en angle Y
        } else if (lastPos.x()!= event->x()) {
            // Moure convenientment la càmera en angle X i/o en angle Y
        }
```

```
    }
```

Fixa't que cal inicialitzar la posició del darrer event en la variable lastPost. Utilitza per això, el mètode `void GLWidget::mousePressEvent(QMouseEvent *event).`

Pas 6. Modificació de les classes Escena i GlWidget per incloure la càmera en primera persona

La càmera en primera persona servirà per a seguir la bola blanca i controlar la direcció de la tirada. Es considera que aquesta càmera mira cap al centre de la bola blanca des d'una posició una mica per darrera de la bola blanca i mira cap al ConjuntBoles inicial. Defineix la *window* fixe segons l'amplada de la taula de billar. Defineix els plans de retallat antero-posteriors de forma que l'observador vegi des de la part posterior de la bola fins al final de la taula.

6.1. Modificació de la classe Escena per afegir la càmera en primera persona.

- Afegir l'atribut de tipus Camera que contingui la informació de la càmea en primera persona
- Modificar els mètodes següents per a que tinguin en compte el cas de la càmera en primera persona en el cas que el paràmetre sigui fals.

```
void iniCamera(bool camGeneral);

void setAnglesCamera(bool camGeneral, float angX, float angY, float angZ);

void setVRPCamera(bool camGeneral, point4 vrp);

void setWindowCamera(bool camGeneral, bool retallat, Capsa2D window);

void setDCamera(bool camGeneral, float d);
```

6.2. Modificació de la classe GlWidget per modificar interactivament la càmera en primera persona

Amb a tecla 'b' es vol canviar la visualització des de la càmera general a la càmera en primera persona. Amb la tecla 't' es canviarà des de la càmera en primera persona a la càmera general.

Com fer-ho?

Canviar la càmera activa a la càmera general o en tercera persona, depenent de la tecla premuda. Això ho pots controlar amb els mètodes:

```
void keyPressEvent(QKeyEvent *event);
void keyReleaseEvent(QKeyEvent *event);
```

6.3. Modificació de la càmera en primera persona amb el moviment de la bola blanca

Quan es mou la bola blanca amb les tecles de les fletxes, es vol que la càmera en primera persona la segueixi. En aquest punt, cal modificar el VRP i l'observador, sense alterar la distància inicial de la càmera.

Com fer-ho?

En el mètode de servei de la interrupció del teclat, caldrà cridar a la modificació de la càmera, convenientment.

2. Planificació:

Les dates límit de lliurament de la pràctica 2 és els dies 27 i 28 d'abril en el teu grup de pràctiques. A continuació es detalla la planificació aconsellada per a desenvolupar-la.

Març	**23**	**24**	**25**	**26**	**27**	**Enunciat de la pràctica 2**
Abril	30	31	1	2	3	Punts 1, 2 i 3 implementats
	6	7	8	9	10	Punt 4 implementat
	13	14	15	16	17	Punt 5 implementat
	20	21	22	23	24	Punt 6 implementat
	27	28	29	30	1	Lliurament de la pràctica 2