Pràctica número 2: Afegint càmeres a les Carreres

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 panoràmica amb Qt i OpenGL utilitzant *shaders*. Navegació en tercera persona per l'escena.

La pràctica té com objectiu afegir una càmera dinàmica controlada des de la interfície per a poder obtenir diferents vistes de l'escena del circuit de carreres. Així mateix es permetrà la navegació per l'escena en tercera persona des del punt de vista del cotxe.

A la pràctica 1 has construït i has visualitzat models poligonals des d'un punt de vista determinat i les rotacions es feien directament sobre els models que formaven part de l'escena (terra, cotxe i obstacles). 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 tercera persona des del punt de vista del cotxe (navegar en tercera persona consisteix en que la càmera segueix en tot moment el cotxe).

L'escena contindrà dues càmeres: **la càmera panoràmica**, que permetrà visualitzar tot el circuit i **la càmera en tercera persona** que seguirà en tot moment el moviment del cotxe. La pràctica 2 es composa de 3 parts:

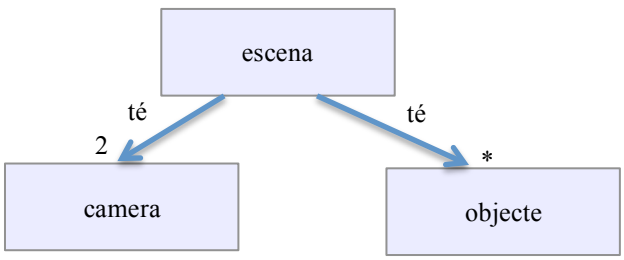
- modificació de la classe **Càmera** (proporcionada en el campus virtual de l'assignatura) per incloure el càlcul de les matrius *model-view* i *projection*, que permetin tant la visualització de l'escena completa com parts d'ella, des de la GPU.
- implementació dels mètodes de la classe **escena** **actualitzaCameraPanoramica(bool clip)** i **resetCameraPanoramica()** que permetin el càlcul dels atributs de la càmera panoràmica per permetre canvis en els angles de visió, canvis de zoom i de *panning* i la inicialització de la càmera panoràmica el primer cop. És a dir, aquests mètodes actualitzen convenientment la càmera panoràmica, segons la interacció de l'usuari.
- implementació dels mètodes **iniLookAtCotxe()** i **actualitzaCameraThirdPerson()** en la classe **escena** que permet el càlcul dels atributs de la càmera associada a la vista en tercera persona.

En aquest document, primer es descriu el programari proporcionat per a començar la pràctica, descrivint l'ampliació que cal realitzar del diagrama de classes i els principals atributs de la classe **càmera**. Després, es detalla com es defineixen els *shaders* per a connectar-los amb el programa i com incloure en el *vertex shader* el codi necessari per a realitzar les transformacions de visualització. Llegeix aquest apartat abans de començar a implementar.

Finalment, es detallen els enunciats dels exercicis que cal desenvolupar en la pràctica 2, que es lliurarà en el campus amb un qüestionari presencial el mateix dia del lliurament de la pràctica, amb la planificació dels mateixos al llarg del temps. També s'inclouen possibles ampliacions que pots animar-te a implementar.

1. Codi inicial de la pràctica 2

Per a desenvolupar la pràctica 2, parteix del codi de la pràctica 1, del qual s'ha de modificar la classe escena amb noves utilitats i s'afegeix la nova classe Camera, segons es detalla en el diagrama de classes següent:



1.1 Classe escena

La classe **escena** inclou tots els elements necessaris per a realitzar la visualització. En aquesta segona pràctica s'inclouen dues càmeres: la **càmera panoràmica** i la **càmera en tercera persona**. En la pràctica 3 s'inclouran les llums, els materials i les textures dels objectes per a donar realisme a la visualització final.

S'afegiran com a mínim dos nous mètodes. Si durant la implementació creus que et calen més mètodes, no dubtis en implementar-los i documentar-los convenientment.

```
void actualitzaCameraPanoramica(bool clip);

void resetCameraPanoràmica();

void actualitzaCameraThirdPerson();

void iniLookAtCotxe();
```

El mètode **actualitzaCameraPanoramica** s'encarrega de donar valors a tots els atributs de la càmera panoràmica que estan relacionats amb la visualització. El paràmetre `clip` indica si es veu part de l'escena o l'escena sencera, és a dir, si s'aplica *clipping* o no. Si és fals, tota l'escena serà visible per l'observador, des de la càmera panoràmica.

El mètode **resetCameraPanoramica** situarà la càmera panoràmica a la seva posició original. Inicialment la càmera panoràmica està mostrant tota l'escena en planta (vista TOP) i l'observador està mirant al centre de l'escena.

El mètode **actualitzaCameraThirdPerson** s'encarrega d'actualitzar la càmera en tercera persona quan es mou el cotxe. Amb el mètode **iniLookAtCotxe** inicialitz la càmera en tercera persona. Aquesta càmera sempre està mirant cap a la part davantera del cotxe des d'una posició de darrera el cotxe i una mica elevada. La càmera en tercera persona sempre mostra una part reduïda de l'escena, centrada en el cotxe.

Tingueu en compte que cal definir els valors de la window i mantenir la coherència dels càlculs de les matrius al final d'aquest mètode (de la matriu model-view i la matriu projection). Cal modificar el codi per a poder utilitzar les matrius model_view i projection des del vertex shader per a poder visualitzar correctament l'escena. Recordeu que si hi han canvis en aquestes matrius, cal tornar-les a passar al vèrtex shader.

En tot moment, només una de les dues càmeres està activa en la interfície. S'ha de tenir en compte que cada vegada que s'afegeix un objecte a l'escena, cal recalcular la càmera activa per a obtenir la visualització de tota l'escena actual des de la càmera panoràmica.

1.2 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 */
 } VisuSystem;

typedef struct
 {
  TipProj    proj;          /* tipus de proj: 0 paral.lela, 1 perspectiva */
  double     d;             /* distancia observador a pla de projecció */
```

```
  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)
```

En aquest punt 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()**).

Els següents mètodes **NO** estan implementats i caldrà desenvolupar-los com a part d'aquesta pràctica:

```
   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*.

```
   void CalculaMatriuModelView();
```

Calcula la matriu model-view (modView) a partir dels atributs de la càmera.

```
   void CalculaMatriuProjection();
```

Calcula la matriu projection (proj) a partir dels atributs de la càmera.

```
   void CalculWindow(Capsa3D);
```

A partir de la Capsa 3D contenidora de l'escena, calcula el *window* que cal definir per a visualitzar tota l'escena.

```
   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.*

```
   void setModelView(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 setProjection(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.*

1.3 Classe glWidget

Des d'aquesta classe es controla la interacció amb l'usuari del widget GL, el moviment del ratolí, el control del teclat, etc. Amb el ratolí es desitja controlar el moviment de la càmera, la possibilitat de fer *zoom* i *panning*. Amb el teclat, a més a més de poder moure el cotxe, amb la tecla "Escape" es podrà canviar de la càmera panoràmica a la càmera en tercera persona.

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.

Els principals mètodes de la classe glWidget que caldrà modificar són:

```
// s'han de modificar i d'implementar totes les funcions controlant les propietats que s'han de canviar
    de la càmera abans de visualitzar de nou

   GLWidget::GLWidget(QWidget *parent): QGLWidget(QGLFormat(QGL::SampleBuffers), parent)

   void initializeGL();
   void paintGL();
   void resizeGL(int width, int height);

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


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

El mètode `mouseMoveEvent` controlarà els events de ratolí, permetent fer zoom, *panning* I moure els angles de posicionament de la càmera panoràmica. A continuació, es mostra un esquelet del codi d'aquest mètode:

```
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
      }
   } else if (event->buttons() & Qt::RightButton) {
      // Panning: moure el centre de la window
      Pan(dx, dy);
   } else if (event->buttons() & Qt::MidButton) {
     // Zoom: canviar la mida de la window un tant per cent
      if(lastPos.y()> event->y())
         Zoom(-1);
      else
         Zoom(1);
   }
   lastPos = event->pos();
}
```

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

2. Connexió amb els *shaders*

2.1. Inicialització dels *shaders* des de l'aplicació

En aquest programa s'usaran només un *shader* pels vèrtexs i un *shader* pels fragments. Per inicialitzar els programes dels *shaders*, des de la classe **escena** s'inicialitzen els *shaders*, carregant el codi i compilant-lo des de l'aplicació. La utilització de *shaders* des del programa permet de carregar-los de forma dinàmica des de l'aplicació en execució, en el cas que canviïn el codi.

Per inicialitzar els *shaders* s'utilitza el mètode **InitShader,** que crea i compila en temps d'execució l'objecte de GLSL a executar-se, a partir dels noms dels fitxers que contenen el vertex shader i el fragment shader.

```
void escena::InitShader(const char* vShaderFile, const char* fShaderFile)
{
   struct Shader {
       const char*  filename;
       GLenum       type;
       GLchar*      source;
    }
    shaders[2] = {
       { vShaderFile, GL_VERTEX_SHADER, NULL },
       { fShaderFile, GL_FRAGMENT_SHADER, NULL }
    };

    QGLShader *vshader = new QGLShader(QGLShader::Vertex, this);
    QGLShader *fshader = new QGLShader(QGLShader::Fragment, this);

   // Es llegeixen els dos shaders: el vertex i el fragment shader
    for ( int i = 0; i < 2; ++i ) {
       Shader& s = shaders[i];
       s.source = Common::readShaderSource( s.filename );
       if ( shaders[i].source == NULL ) {
          std::cerr << "Failed to read " << s.filename << std::endl;
          exit( EXIT_FAILURE );
       }
   }
    // Es compilen els programes en temps d'execució de l'aplicació

    vshader->compileSourceCode(shaders[0].source);
    fshader->compileSourceCode(shaders[1].source);

  // S'afegeixen a una variable de classe
    program = new QGLShaderProgram(this);
    program->addShader(vshader);
    program->addShader(fshader);

  // Es munta el programa
    program->link();

  // Es vincula el programa al context de GL per a ser executat amb les comandes de GL
  program->bind();
}
```

2.2. Inclusió de les transformacions model-view i projection en el vertex shader

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*.

Per a poder usar-les 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):

PAS 1: Definir les variables a la CPU i a la GPU (en el vertex shader)

En la CPU:

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

En la GPU (en el vertex shader):

```
uniform mat4 model_view;
```

PAS 2: Passar la informació de la matriu de la CPU a la GPU, des del programa:

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

PAS 3: Utilitzar-la en el vertex shader:

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

3. Exercicis a implementar

Abans de començar, baixa del campus virtual la classe càmera (càmera.cpp i càmera.h) i afegeix la classe al teu projecte. Baixa també la classe mat.h i substitueix-la per la que tens ara.

3.1. Implementació 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.

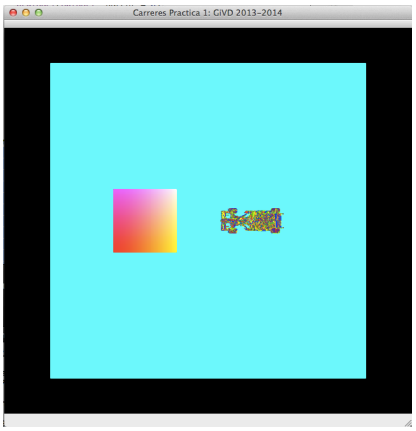
3.2. Modificació de la classe Càmera: Cal implementar en la classe càmera els mètodes d'aquesta classe detallats en l'apartat 1.2. Comença implementant el mètode `toGPU()` de la classe càmera. Implementa ara els mètodes `ini()`, `CalculaMatriuModelView()`, `CalculaMatriuProjection()` i `CalculWindow()`. 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()`.

Els mètodes `CalculaMatriuModelView()` i `CalculaMatriuProjection()` calculen les matrius model-view i projection utilitzant els mètodes proporcionats en el fitxer mat.h.

El mètode `CalculWindow()` defineix els atributs de la window per a que es pugui veure tota l'escena en projecció paral·lela.

3.3. Implementació dels mètodes resetCameraPanoramica i actualitzaCameraPanoramica a la classe Escena:

En la classe Escena cal implementar el mètode **`actualitzaCameraPanoramica`**`(bool clip)` que permetrà calcular atributs de la càmera panoràmica, segons si es vol visualitzar o no tota l'escena (paràmetre de `clip`).



Per a desenvolupar de forma incremental aquest mètode, afegeix un atribut **cameraActual** a la classe glWidget, inclou la creació de la càmera panoràmica quan crees una escena i 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 tres apartats anteriors, hauries de veure el circuit de carreres sencer en una vista de planta (TOP).

3.4. Interacció amb la càmera panoràmica i l'escena: Modificació de la classe glWidget

A partir de la interacció amb el ratolí es poden realitzar diferents modificacions de la càmera: (mira l'esquelet del mètode `mouseMoveEvent` detallat en l'apartat 1.3 d'aquest enunciat)

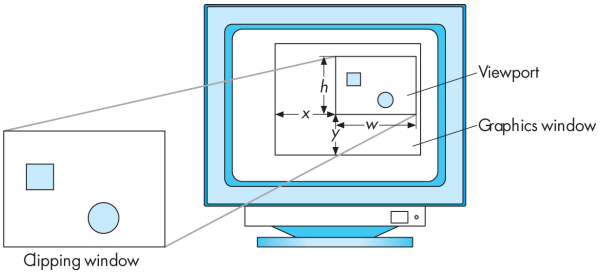
- arrossegant el ratolí amb el **botó esquerra** pitjat es canvien els **angles** de visió de la càmera, segons el desplaçament. Si hi ha un desplaçament en X de viewport, es canviarà l'angle Y de la càmera i si hi ha un desplaçament en Y de viewport es canviarà l'angle X de la càmera.

- arrossegant el ratolí amb el **botó del mig** pitjat es farà un **zoom** de l'escena. Si es fa un desplaçament positiu en coordenades Y de viewport, es farà una ampliació de la visualització amb un factor d'escala fix (per exemple de 0.05). Si es fa un desplaçament negatiu en coordenades de Y de viewport, es farà una reducció de la visualització amb el mateix factor d'escala.

- arrossegant el ratolí amb el **botó de la dreta** pitjat 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.

Per a realitzar aquest apartat, implementa els mètodes de la classe **glWidget** `Zoom`, `Pan`, `setXRotation`, `setYRotation`. Basa't en les accions que has desenvolupat pel càlcul de la càmera i en el mètode `actualitzaCameraPanoramica`, que has implementat prèviament.

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



3.5. Navegació de l'escena en tercera persona

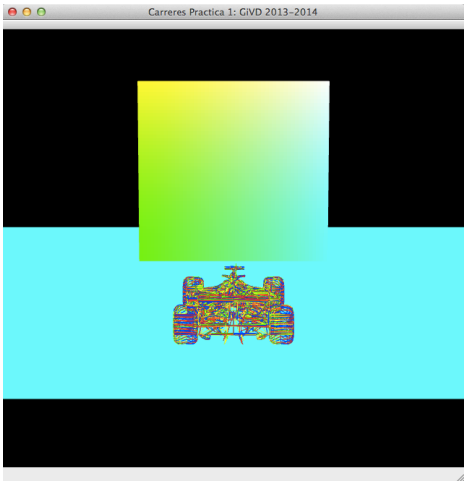
3.5.1 Implementació del mètode iniLookAtCotxe() a la classe Escena:

El mètode `iniLookAtCotxe` inicialitza els atributs de la càmera en tercera persona, un cop s'ha llegit el cotxe. Es considera que aquesta càmera mira a la part davantera del cotxe des d'una posició una mica per darrera del cotxe i a una certa alçada. Defineix la *window* fixe segons la dimensió proporcional a la capsa

frontal del cotxe inicial. Defineix els plans de retallat antero-posteriors de forma que l'observador vegi des de la part posterior del cotxe fins al final del circuit.

3.5.2. Implementa el mètode actualitzaCameraThirdPerson() a la classe Escena per a controlar la càmera en tercera persona:

Implementa el mètode `actualitzaCameraThirdPerson()` per a que actualitzi la càmera en tercera persona, si és necessari. Afegeix la càmera en tercera persona a l'escena. Canvia el mètode keyPressEvent per a que tingui en compte l'event de la tecla "Escape" (Qt::Key_Escape). En aquest punt obtindràs una visualització semblant a aquesta.



3.5.3. Actualització de la càmera en tercera persona amb el moviment del cotxe

Modifica el mètode `keyPressEvent` per a controlar la càmera en tercera persona quan es mou el cotxe. En aquest punt, es mouran el VRP i l'observador, sense alterar-se la distància inicial de la càmera. Recalcula les matrius que consideris necessàries a cada pas, usant el mètode `actualitzaCameraThirdPerson`() implementat a l'apartat anterior.

4. Planificació:

El lliurament de la pràctica 2 és els dies 28 i 29 d'abril en el teu grup de pràctiques. A continuació es detalla la planificació aconsellada per a desenvolupar-la.

Març	24	25	26	27	28	Enunciat de la pràctica 2
Abril	31	1	2	3	4	Punts 3.1, 3.2 i 3.3 implementats
	7	8	9	10	11	Punt 3.4 implementat
	14	15	16	17	18	Punt 3.5 implementat
	21	22	23	24	25	Proves finals
	28	29	30	1	2	Lliurament de la pràctica 2

5. Extensions addicionals: (OPCIONAL)

5.1. Inclusió dels càlculs del moviment del cotxe a la GPU.

Per accelerar el càlcul de les transformacions geomètriques que s'han d'aplicar al cotxe, passa la matriu de TG a la GPU, per a que es calculin en la GPU les rotacions i/o translacions aplicades a cada part del cotxe. Defineix una matriu que acumuli les TGs aplicades a un objecte i aplica-la a cadascun dels seus punt en la GPU. Canvia la crida aplicaTG per aplicaTG_GPU.

5.2. Jugant amb dos cotxes

Si has desenvolupat l'extensió de la primera pràctica d'afegir un segon jugador, afegeix la possibilitat de veure en tercera persona el segon jugador, quan es torni a prémer la tecla "Escape".

5.3. Visualització simultània amb la càmera panoràmica i la càmera en tercera persona

Defineix un nou glWidget en la teva interfície gràfica per tal de veure la visualització panoràmica en el vell glWidget i la visualització en tercera persona en el nouWidget.