GiVD - curs 2014-2015 – Pràctica 1

Pràctica 1: Modelat, visualització d'un joc de billar

Objectiu: Modelat, visualització i interacció amb una escena que simula un *d'un joc de billar* amb Qt i OpenGL utilitzant *shaders*.

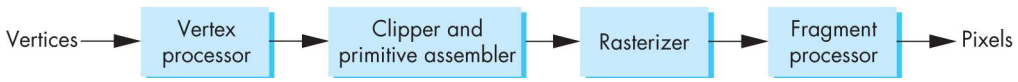
En aquesta pràctica, modelareu i visualitzareu una escena virtual d'un joc de billar americà (bola blanca i 15 boles), incloent una interacció senzilla de joc.

Al final de la pràctica 1, tindreu una escena de taula i boles de billar on es podrà moure la bola blanca amb les fletxes de teclat i simular el tir de la bola blanca. La única bola que el jugador podrà moure és la bola blanca i quan aquesta bola col·lisioni amb altres boles o amb els límits de la taula, només rebotarà, sense produir moviment a les altres boles. Les altres boles, en aquesta fase, no es mouran (actuaran com obstacles).

Aquesta pràctica té com objectius:

- Aprendre el modelatge d'una escena virtual,
- Aprendre la programació de Transformacions Geomètriques sobre objectes 3D,
- Aprendre la programació de la interacció de l'usuari amb l'escena,
- Aprendre com visualitzar les textures de cadascuna de les boles de billar.
- Conèixer els mecanismes de OpenGL i GLSL per a visualitzar escenes.

La pràctica es desenvoluparà a partir del codi del campus (BillarPractica1.tgz) que és l'esquelet buit de la pràctica amb els menú QT ja creats. Recorda l'arquitectura bàsica de l'aplicació, que utilitza els programes *vertex shader* i *fragment shader,* que re-programen la part del *pipeline* d'OpenGL referent a les transformacions dels vèrtexs i els càlculs del color (repassa la practica0 si en tens dubtes).



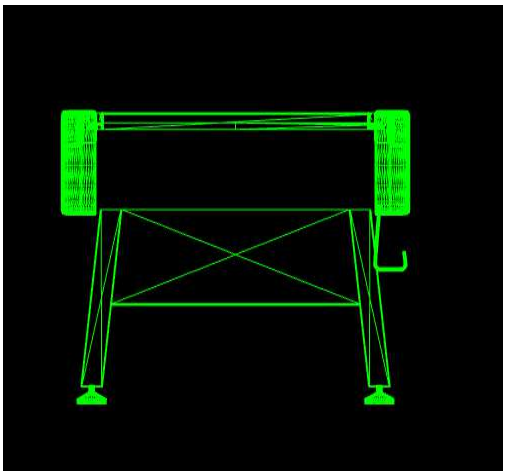
El lliurament de la pràctica es realitzarà en el campus virtual i tindrà associat un qüestionari que s'ha de realitzar de forma presencial sobre el codi desenvolupat després del seu lliurament. En el Campus Virtual trobaràs les instruccions de com lliurar una pràctica a l'assignatura.

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 un joc més real.

1 Etapes de desenvolupament de la pràctica

1. **Instal·lació:** Descarrega la pràctica1 del Campus Virtual (BillarPractica1.tgz), executa-la en l'entorn de QtCreator i mira de carregar la taula de billar del fitxer taula.obj de la carpeta resources. En aquesta primera execució, es visualitza la taula centrada i vista des de davant per que en el codi es fa manualment la

transformació geomètrica necessària per a que es visualitzi centrada i sense retallar.



Per a obtenir un primer model de la taula de billar, es llegeix el fitxer taula.obj de la carpeta resources. El format .obj és un format ASCII per objectes gràfics més o menys estàndard. Mireu el software gràfic anomenat ***blender*** si voleu modelar nous models www.blender.org. Podeu obtenir més informació sobre el format de fitxer .obj en l'enllaç http://en.wikipedia.org/wiki/Wavefront_.obj_file.

2. Exploració del codi:

L'arquitectura de projecte es basa en un disseny orientat a objectes que es detalla a continuació. Les principals classes son:

main: conté el programa principal de l'aplicació.

mainwindow: conté el *frame* principal de Qt que crea i controla els menús i el *widget* de *GL* on es desenvoluparà l'aplicació.

glwidget: (*és la classe `GLWidget` de la pràctica 0 modificat*) aquesta classe controla els events i la visualització de l'escena. **Cal modificar-la en aquesta pràctica.** Cal tenir en compte que el *`GlWidget`* es basa en una càmera fixe i que visualitza un món centrat en el punt (0,0,0) i que està limitat per la capsa definida entre els punts (-1, -1, -1) i (1, 1, 1). En aquesta classe es tenen els mètodes *newPlaBase*( ) , *newBola(), newConjuntBoles*( ) , *newObjecte*(), *newSalaBillar*() i *Play*() que són cridats des de la interfície del joc. També conté els mètodes de servei de les interrupcions de ratolí *mousePressEvent*() i de teclat *keyPressEvent*( ), així com alguns mètodes auxiliars com *adaptaObjecteTamanyWidget*().

Escena: aquesta classe tindrà tots els objectes que formin el joc final i s'ampliarà en les pràctiques següents. Inicialment, l'escena es considera definida en la capsa que va des del punt (-25, -25, -25) fins al punt (25, 25, 25). **Cal modificar-la i acabar d'implementar-la en aquesta pràctica.**

Els mètodes principals associats a l'escena, que es faran servir des del *`GLWidget`* i que heu d'implementar són:

```
void addObjecte(Objecte *obj); // Afegeix un objecte a l'escena
```

```
void aplicaTGCentrat(mat4 m);     // Aplica una TG a tots els objectes de l'escena
                                  // centrada al punt mig de la capsa mínima
                                  // contenidora de tots els objectes de l'escena

void draw();                      // Visualització de tots els objectes de
                                  // l'escena
void CapsaMinCont3DEscena();      // Capsa contenidora de tots els objectes de
                                  // l'escena
```

Objecte: classe que defineix la interfície de tot model que s'inclou en l'escena. **Cal acabar-la d'implementar en aquesta pràctica**. Es defineixen els mètodes *draw*( ) , *make*( ) , *toGPU*(QGLShaderProgram *program) , *calculCapsa3D*(), *aplicaTG(mat4 m), Centrat(mat4 m)*. Els seus atributs principals, que heretaran tots els tipus d'objectes es poden agrupar en diferents blocs:

- La part de modelatge: el model de l'objecte es basa en la representació cares-vèrtexs, sense tenir repetició de vèrtexs.

```
QString nom;             // nom del fitxer on esta la taula
vector<Cara> cares;      // cares de l'objecte
vector<point4> vertexs; // vertexs de l'objecte sense repetits
```

- La posició i l'orientació dins del món global es defineixen amb un sistema de coordenades local, definit per un punt i tres angles associats als tres eixos. També es defineix la capsa mínima contenidora basada en el sistema de coordenades global:

```
// Sistema de coordenades d'un objecte: punt origen i eixos de rotació
 GLdouble xorig, yorig, zorig;
 double xRot;
 double yRot;
 double zRot;
 GLfloat tam; // Escala de l'objecte aplicada al fitxer d'entrada
 Capsa3D capsa;
```

- Addicionalment es defineixen els atributs necessaris per passar el model a la GPU. Per això s'utilitza un vector de punts i els colors associats a cada punt. L'atribut `Index` porta el control del número de vèrtexs en els vectors de punts i colors. L'atribut `buffer` serveix per guardar la direcció de memòria de la GPU on s'emmagatzemen les dades quan es passen a la GPU amb el mètode *toGPU*().

```
// Programa de shaders de la GPU
QGLShaderProgram *program;
GLuint buffer; // Buffer de comunicacio amb la GPU

// Estructures de vertexs i colors per passar a la GPU
int const numPoints;
point4 *points;
color4 *colors;
int Index;    // index de control del numero de vertexs a posar a
              // la GPU
```

taulaBillar: classe que defineix la taula de billar. Conté la variable `numverticesF` per a dimensionar el vector de punts. Observa que en la constructora hi ha un càlcul per a calcular la transformació geomètrica per a que la taula de billar es pugui veure a la primera visualització. Intenta raonar quina transformació està fent. **Aquesta classe s'haurà de modificar.**

mat.cpp, vec.cpp i **Common.h**: són les classes d'utilitats bàsiques referents a

matrius, vectors i definicions comunes, respectivament. Disposes de diferents tipus bàsics ja implementats, com vec4 que representa una taula de quatre dimensions, *vec3*, *vec2*, *mat4* (matriu de 4x4), *mat3*, *mat2*, etc. Així com el tipus *Capsa3D*.

Readfile: Classe que conté el codi per a llegir un objecte des d'un fitxer .obj.

3. **Afegir un pla base de proves:** Per a fer els primers tests del teu codi i no carregar cada vegada el fitxer de la taula de billar quan facis proves, defineix una nova classe de test que s'anomena **PlaBase,** filla de la classe *Objecte*. Defineix i implementa primer els atributs i les seves constants. Modifica la classe *Escena* per a incloure-li un pla base.

   Cal crear-lo des del menú de Model: Afegeix Pla Base. Per això, implementa el mètode *newPlaBase*() de la classe *GLWidget*. Aquest mètode construirà un pla base, l'afegirà a l'escena com un nou objecte i es visualitzarà. Utilitza el mètode *newObjecte*() de la classe *GLWigdet*. Fes una primera prova per a veure'l, definint una dimensió inferior a 2. Si el defineixes més gran segurament no el veuràs sencer.

   Per a definir el pla base de la taula es considera que:
   - El polígon del pla base està a sobre un pla paral·lel al pla Y = 0, orientat sempre cap a les Y positives.
   - El polígon del pla base és un rectangle que ha d'estar inclòs en l'escena formada entre el punt (-1, -1, -1) i (1, 1, 1).
   - El pla base pot ser un rectangle format per dos triangles. Els atributs i constants deurien permetre guardar:
     - Els punts que formen el pla final
     - Els colors associats a cadascun dels punts

   Per a generar la geometria i la topologia del model del pla utilitzarem l'algorisme de construcció d'una cara del cub (podeu veure el codi del projecte CubGPUTextures.tgz).

   En aquesta visualització, la càmera està situada fixe en l'eix Z=0. Per aconseguir veure el pla base haureu de rotar el seus punts. Cal que utilitzeu el codi de l'event del Ratolí, que acaba cridant al mètode *aplicaTGcentrat*() de la classe *Objecte*. Implementeu aquest mètode i comproveu que funciona primer amb la taula de billar i després amb el pla que acabeu de crear.

4. **Afegeix una nova classe anomenada Bola:** Aquesta nova classe serà filla de la classe *Objecte* i contindrà tots els atributs necessaris per a definir una bola de billar. La classe *Escena* contindrà un objecte de tipus *Bola* que serà la bola blanca i un conjunt de boles que seran les 15 boles restants del joc.

   Per a generar la geometria d'una bola, seguiu l'algorisme recursiu de construcció de la superfície d'una esfera. Aquest algorisme està explicat també en el llibre bàsic de l'assignatura en el tema 5, apartat 5.6. Cal crear la bola des del menú d e Model: Afegeix Bola Blanca, que crida al mètode *newBola*() de la classe *GLWidget*.

   El model de la frontera d'una bola de billar és un model superficial que es modelarà usant una esfera de radi 1, centrada en el punt (0,0,0).

El punt de partida és un tetraedre format per a quatre vèrtexs (0,0,1), (0,2√2/3, -1/3), (√6/3, -√2/3, -1/3) i (√6/3, -√2/3, -1/3), que estan en una esfera de radi 1 i centrada a l'origen. Així, es comença amb els 4 punts:

```
point4 v[4] = {
        vec4( 0.0, 0.0, 1.0, 1.0 ),
        vec4( 0.0, 0.942809, -0.333333, 1.0 ),
        vec4( -0.816497, -0.471405, -0.333333, 1.0 ),
        vec4( 0.816497, -0.471405, -0.333333, 1.0 )
};
```

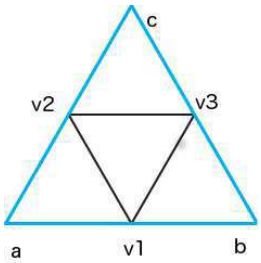
Després, es generen 4 triangles que tenen com extrems aquests vèrtexs, usant la funció triangle que posa en el vector "points" els punts de forma ordenada:

```
void Bola::triangle( const point4& a, const point4& b, const point4& c )
        {
        points[Index] = a;
        Index++;
        points[Index] = b;
        Index++;
        points[Index] = c;
        Index++;
        }
```

i es genera el tetraedre fent 4 crides a la funció triangle:

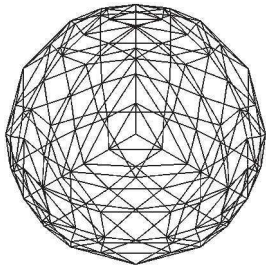
```
triangle( v[0], v[1], v[2]);
triangle( v[3], v[2], v[1]);
triangle( v[0], v[3], v[1]);
triangle( v[0], v[2], v[3]);
```

Per dividir ara el tetraedre s'afegeixen 4 triangles a cadascun dels triangles generats. Per això es poden fer diferents estratègies. Per a generar tres triangles equilàters a partir d'un triangle equilàter (format pels punts a, b i c), s'afegeixen els tres punts mitjos de les arestes del triangle original:



Aquests nous punts, però, estan en el mateix pla del triangle. La idea es moure aquests punts a la superfície de l'esfera normalitzant els nous vèrtexs generats. Llavors, es subdivideix el triangle fent les següents crides:

```
point4 v1 = calculVectorUnitari( a + b );
point4 v2 = calculVectorUnitari( a + c );
point4 v3 = calculVectorUnitari( b + c );
divide_triangle(a, v1, v2);
divide_triangle(c, v2, v3);
divide_triangle(b, v3, v1);
divide_triangle(v1, v3, v2);
```



Ara només, hem d'aplicar aquest codi a totes les cares del tetraedre inicial, i de forma recursiva, aplicar-lo a cadascuna de les noves cares generades, fins a un límit d'iteracions (`NumIteracionsEsfera`).

Implementa aquest codi i prova'l per a veure una esfera en pantalla. Quan la tinguis generada i ben visualitzada, caldrà que l'escalis i posicionis en la taula de billar carregada des de fitxer. Mira les dimensions de la capsa 3D de la taula de billar. Et poden guiar sobre l'escalat i la posició. El taulell on es juga està en el pla Y=0. Fes unes primeres proves de test amb el pla base per a veure que la posiciones bé l'esfera en el pla.

5. **Adapta la mida de l'escena al widget de defecte:** Addicionalment, per tal que es visualitzi en el món creat en la classe *`GLWidget`* es visualitza només tot el que està contingut en el cub (-1, -1, -1) – (1, 1, 1). Per a visualitzar l'escena completa, implementa en la classe *`GLWidget`*, el mètode:

   ```
   void GLWidget::adaptaObjecteTamanyWidget(Objecte *obj)
   ```

   Suposa que l'escena completa està inclosa en una capsa de 20x20x20 centrada en el (0,0,0). Aquest mètode permetrà traslladar i escalar convenientment els objectes continguts en l'escena en el *`GLWidget`* per defecte (que suposa que el *fustrum* és un cub que té l'origen al punt (-1, -1, -1) i dimensió 2 i la projecció és paral·lela). Considera per tal d'accelerar càlculs, la possibilitat de tenir pre-calculades alguna de les matrius que s'han de fer servir per aquesta transformació.

6. **Inclusió de la classe que conté el conjunt de boles de colors:** Inclou una nova classe *`ConjuntBoles`* al projecte, que utilitzaran l'algorisme recursiu de construcció d'una esfera per a cada bola. Inclou un atribut a l'escena que contingui aquest conjunt.

En la seva construcció, situa les boles i fixa la seva mida en l'escena segons la posició inicial del billar.

Ajuda't del pla base per a situar-les abans de fer-ho a la taula sencera. Escala el teu pla Base a la mateixa mida del taulell de la taula del fitxer. Fixa't en la Capsa contenidora que es calcula quan es carrega el fitxer per conèixer les mides.

Cal crear el conjunt de Boles invocant el menú Model: Afegeix 15 Boles, que crida al mètode *newConjuntBoles*() de la classe *`GLWidget`*.

7. **Carrega tota l'escena sencera:** En aquest pas es suposa que la classe **Escena** guardarà tots els objectes del joc: la bola blanca, les boles de colors i la taula de billar. Cal crear tota l'escenari invocant el menú Model: Afegeix Sala Billar, que crida al mètode *newSalaBillar*() de la classe *`GLWidget`*.

8. **Afegeix realisme a l'escena:** Incorporant textures a les boles de colors. Cal que modifiques les classes *Bola* i *ConjuntBoles* per a poder treballar amb textures. Recorda que s'han de carregar les textures (disposes de les textures de cadascuna de les boles en la carpeta de resources), generar les coordenades de textura per a cada punt de la bola i passar-les al shader.

   Per a generar les coordenades de textures associades a cada vèrtex de l'esfera, es pot utilitzar la següent fórmula, tot suposant que x, y, z és el vector unitari que va des del punt p de la superfície de l'esfera fins al del centre de l'esfera.

   $u = 0.5 + \arctan2(z, x)/ 2\Pi$
   $v = 0.5 - \arcsin(y)/ \Pi$

   Recorda que (u,v) son valors entre 0 i 1.

9. **Afegeix dinamisme a l'escena**, incloent la interacció amb l'usuari. Amb les tecles del teclat es desitja dotar d'increments de moviment a la bola blanca. Per això:

   - Sobrecarrega els mètodes *KeyPressEvent()* i *KeyReleaseEvent()* per a realitzar la Transformació Geomètrica corresponent al moviment de la bola blanca. Amb les tecles de les fletxes es simula un desplaçament de valor `deltaDesplaçament` a l'esquerra, avall, a la dreta i amunt de la bola blanca.
   - La bola es mou sempre sobre el pla de la taula (pla Y) i no pot excedir els límits del tauler.
   - Realitza el control de col·lisions entre la bola blanca i la resta de boles (en aquesta primera pràctica la única bola que es mou és la bola blanca controlada des del teclat per l'usuari).

2 Extensions opcionals

Extensió 1: Per afegir més **realisme** al moviment, pots considerar que la tecla de l'espai llança la bola blanca provocant l'animació continuada de la bola. Per això necessites refrescar constantment l'escena amb un *Timer* (mira el projecte CubGL.tgz on se n'usa un). Has de controlar la velocitat de la bola i calcular el seu desplaçament a cada event del *Timer*.

Quan més temps es tingui premuda la tecla de l'espai, més ran serà la força inicial d'impuls que es donarà a la bola. A partir de la primera velocitat inicial, la roda comença a reduir la seva velocitat a causa del fregament fins a arribar a zero. La bola blanca ha de desplaçar-se al llarg d'una recta i rodar sobre si mateixa al voltant d'un eix perpendicular a la recta. Considera que la recta iniciar és paral·lela a l'eix Z i per tant, l'eix de rotacio de la bola és l'eix X.

Des del Menú Controls: Play activa l'opció d'aquesta extensió. Implementa-la en el mètode *Play()* de la classe *GLWidget*.

Extensió 2: Lectura d'una taula formada per varies peces. En el codi de la pràctica inicial es disposa d'un mètode que llegeix un fitxer .obj (*readObj*() de la classe *Objecte*). La lectura de l'objecte es fa directament en el model cares-vèrtexs. Es per aquest motiu que la classe *Objecte* utilitza la classe *Cara* per a definir la llista de

cares.

El mètode *readObj*() de la pràctica inicial suposa que en el fitxer només hi ha un objecte, encara que el fitxer conté diferents parts. Cada part comença amb una línia on el seu primer caràcter és la lletra "o". Si obriu el fitxer TaulaParts.obj amb un editor de texte, podreu explorar més fàcilment el seu contingut.

Modifica el mètode *readObj*() per permetre la lectura de varies parts de la taula de billar (les potes, el taulell de joc i cadascun dels sis forats de la taula). Modifica la classe *TaulaBillar* per a que el codi permeti la lectura de totes les parts de la taula per fitxer.

Modifica les col·lisions de la bola blanca per a saber si la bola ha xocat contra un dels forats. Si fos així torna a colocar la bola blanca en la seva posició inicial al següent frame.