GiVD - curs 2013-2014 – Pràctica 1

Pràctica 1

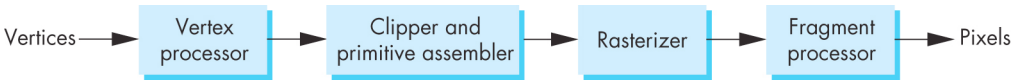
Tema: Modelat, visualització d'un joc de carreres de cotxes
Objectiu: Modelat, visualització i interacció amb una escena que simula un *circuit de carreres de cotxes* amb Qt i OpenGL utilitzant *shaders*.

Aquesta pràctica té com objectius:

- aprendre el modelatge d'una escena virtual,
- la programació de Transformacions Geomètriques sobre objectes 3D,
- la programació de la interacció de l'usuari amb l'escena,
- el control de col·lisions entre objectes

Per a assolir aquests objectius, modelareu i visualitzareu una escena virtual d'un circuit de carreres, incloent la interacció d'un joc . El circuit inclou un terra, un cotxe format per diferents parts (carrosseria i rodes) i obstacles (que inicialment seran cubs). Caldrà que realitzeu transformacions geomètriques que permeten passar del sistema de coordenades local d'un objecte a coordenades globals de món i visualitzareu el circuit sobre l'arquitectura de OpenGL i GLSL. Així mateix li donareu la interacció amb les fletxes del teclat que permetin moure el cotxe. Addicionalment, es podran introduir obstacles i fer el control de col·lisions de forma automàtica.

La pràctica es desenvoluparà a partir del codi del campus (Carreres.tgz) que és l'esquelet buit de la pràctica amb els menú 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.



El lliurament de la pràctica es realitzarà en el campus virtual i tindrà associat un qüestionari sobre el codi desenvolupat després del seu lliurament.

Aquest enunciat es composa d'una primera part on s'expliquen diferents aspectes implicats en la pràctica. Després, es detallen els passos a seguir en el desenvolupament de la pràctica. Finalment, es proposen extensions opcionals que es poden implementar.

1 Explicació de la pràctica

A continuació, es detallen diferents aspectes implicats en la pràctica, que us poden servir de guia per a la seva realització.

- explicació del modelat del circuit
- explicació de la construcció d'una nova classe que modela un cotxe amb les seves diferents parts (carrosseria i rodes) i els seus moviments: definició de la geometria del cotxe, translació a partir del seu eix principal i rotació sobre el seu eix.

- explicació de la inclusió en el món global (escena) i la seva visualització d'OpenGL i *shaders*
- explicació de la interacció amb el teclat.

1.1. Modelat del circuit (Terra i Obstacles)

Per a definir el terra del circuit es considera que:

1. El polígon del circuit està a sobre un pla paral·lel al pla Y = 0, orientat sempre cap a les Y positives.
2. El polígon del circuit és un rectangle que ha d'estar inclòs en l'escena (l'escena estarà definida dins de cub centrat en el punt (0,0,0) i d'aresta 50.
3. Les dimensions i la Y per un passa el pla del terra es defineixen en una finestra de diàleg ja implementada.
4. El terra pot ser un rectangle format per dos triangles. Els atributs i constants deurien permetre guardar:
    - Els punts que formen el terra 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 CubGPU.tgz).

5. En el circuit hi han obstacles. En aquesta part els obstacles són cubs repartits de forma aleatòria pel circuit.

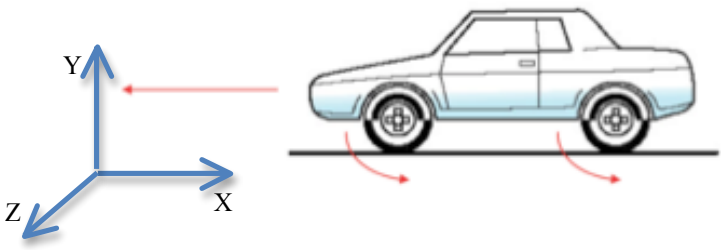
Per definir els obstacles del circuit es suposa:

1. Els obstacles inicialment són cubs
2. La posició i les dimensions dels cub són aleatòries, encara que sempre ha de quedar la base del cub en la Y del pla que defineix el terra.
3. Per a la construcció del cub es partirà de la construcció d'un cub prototipus d'aresta 1, centrat en el punt (0,0,0), al que s'ha d'aplicar les transformacions geomètriques necessàries per a traslladar-los i escalar-los convenientment.
4. El número d'obstacles es demana des d'una finestra de diàleg.

Com a extensió, els cubs també es podran rotar aleatòriament i es podran substituir els cubs per altres tipus d'objectes (veure l'apartat final de l'enunciat).

1.2. Modelat del cotxe

El model de cotxe en aquest circuit és un cotxe de Fórmula 1, que està format per un conjunt de triangles. El model de base és la representació basada en cares-vèrtexs, on no es guarden vèrtexs repetits. Pel cas del cotxe, es vol guardar també com a atribut la direcció d'avançada. En el cas de la figura, el vector d'avançada del cotxe és el vector (-1.0, 0.0, 0.0).



Suposarem que un cotxe està format per la carrosseria i les seves quatre rodes, per a poder-li aplicar diferents transformacions geomètriques durant el seu moviment per la pista. Per exemple, quan el cotxe va endavant, la carrosseria s'ha de traslladar i les rodes cal traslladar-les i girar-les.

En la pràctica base, es considera que el cotxe és un únic objecte i s'escala a dimensió 1, situant-lo en una certa posició de l'escena per tal d'obtenir aquesta primera visualització. En la pràctica, cal dissenyar les classes corresponents per a modelar i controlar el tipus de transformació a cadascuna de les parts del cotxe. També s'hauran de fer les Transformacions Geomètriques per posicionar-lo i donar-li la mida que entri l'usuari.

Per a obtenir un primer model de cotxe, es llegeix el fitxer **Ferrari.obj**. 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.

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 cara-vèrtex. Es per aquest motiu que la classe Objecte utilitza la classe **cara.cpp** per a definir la llista de cares.

Quan s'utilitza el menú de lectura del cotxe, es demanen la posició (en x i z) del cotxe en el circuit, tot suposant que la posició de les Y's de la base del cotxe es situaran a la Y on s'ha definit el terra del circuit. A més a més, es demana a l'usuari l'escala i l'orientació d'avançada en la que està posicionat el cotxe, ja que és una informació que no està continguda en el fitxer .obj.

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 Ferrari.obj amb un editor, podreu explorar més fàcilment el seu contingut.

1.3. Disseny de classes:

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

- **main.cpp:** conté el programa principal de l'aplicació.

- **mainwindow.cpp:** conté el *frame* principal de Qt que crea i controla els menús i el *widget* de *GL* on es desenvoluparà l'aplicació. **Cal modificar-la en aquesta pràctica.**

- **glwidget.cpp:** (*és la classe GLWidget de la pràctica 0 modificat*): aquesta classe controla els events i la visualització de l'escena del circuit i del cotxe. **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 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 *newTerra*(), *newObstacle*() i *newCotxe*() que reben els valors introduïts en les finestres de diàleg.

- **escena.cpp:** aquesta classe tindrà tots els objectes que formin el circuit 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 aplicaTG(mat4 m);        // Aplica una TG a tots els objectes de 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.cpp**: 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), aplicaTGCentrat(mat4 m)*. Els seus atributs principals, que heretaran tots els tipus d'objecteses poden agrupar en diferents blocs:

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

```
QString nom;            // nom del fitxer on esta el cotxe
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
```

  - 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;
```

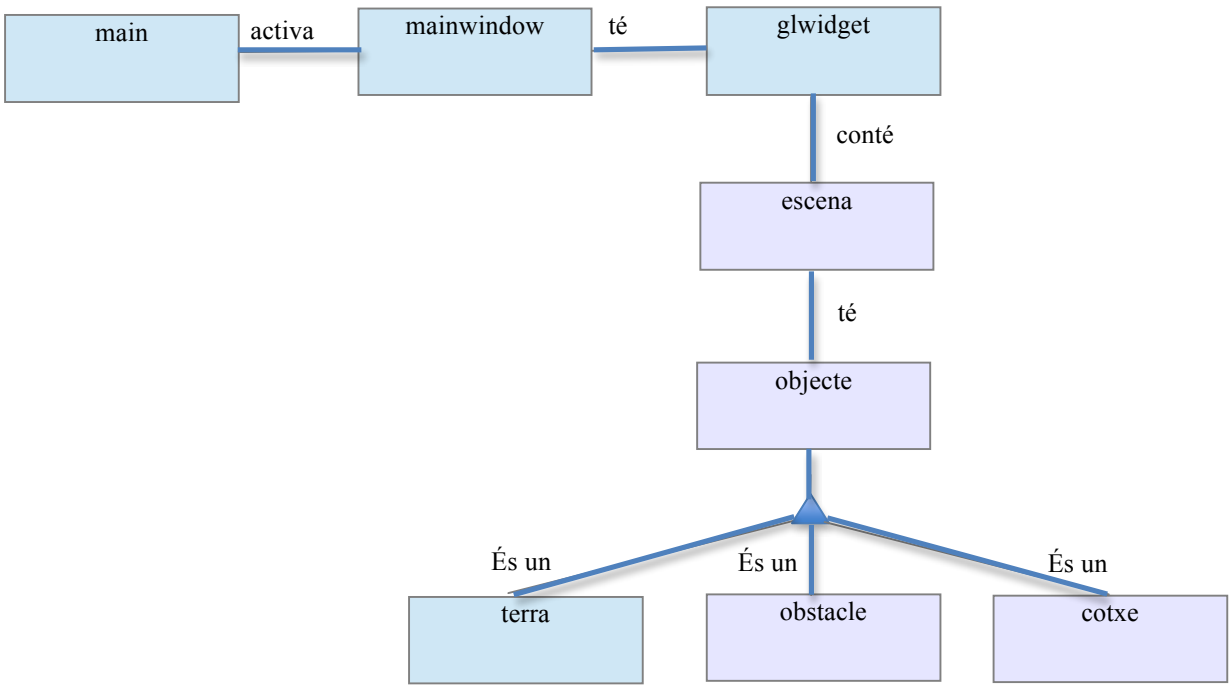
```
int Index;     // index de control del numero de vertexs a posar a
               // la GPU
```

- **obstacles.cpp, terra.cpp**: classes filles d'Objecte que implementen els obstacles i la terra. En aquestes classes s'implementaran els mètodes virtuals que calguin de la classe objecte per a instanciar-los com un cub o com un polígon de terra. **Cal implementar-es en aquesta pràctica, així com afegir els atributs necessaris a la classe escena per a que disposi d'un terra i una llista d'obstacles.**

- **cotxe.cpp**: Es la classe que modela un cotxe. Cal ampliar-la per a que el cotxe estigui format per varies parts: la carrosseria, la roda davantera esquerra, la roda davantera dreta, la roda posterior esquerra i la roda posterior dreta. Cal que sobrecarregueu el mètode *readObj()* de la classe Objecte per a que pugui carregar els diferents objectes que formen el cotxe. Per a fer les proves, disposeu del fitxer anomenat **ferrari.obj** que modela un cotxe de Fórmula 1 en diferents parts. **Cal acabar d'implementar la classe cotxe.cpp en aquesta pràctica**. **Cal també implementar les classes roda.cpp i carrosseria.cpp. Cal modificar la classe escena per poder manegar diferents cotxes.**

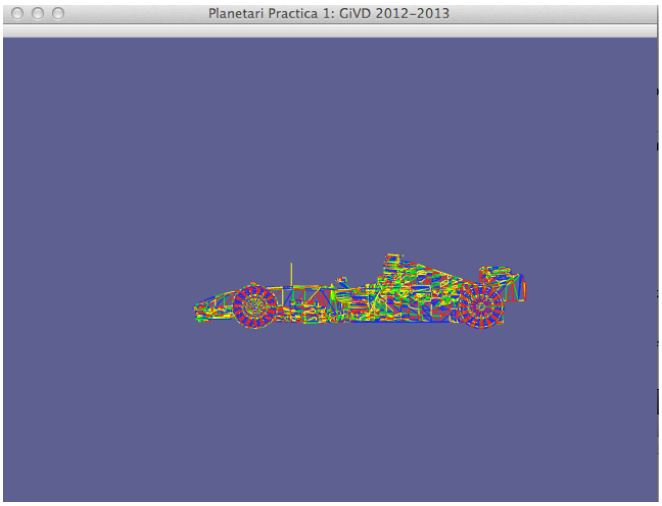
- **mat.cpp, vec.cpp** i **Common.h**: són les classes d'utilitats bàsiques referents a matrius, vectors i definicions comunes, respectivament. Disposes de diferents tipus bàsics ja implementats, com vec4 que representa una taula de quatre dimensions, vec3, vec2, mat4 (matriu de 4x4),mat3, mat2, etc. Així com el tipus Capsa3D.

El diagrama de les classes descrites anteriorment es mostra a continuació, encara que caldrà que s'ampliï amb noves classes en el moment que es defineixi el cotxe i les seves parts:



2 Desenvolupament de la pràctica

1. Descarrega la pràctica1 del Campus Virtual (Carreres.tgz), executa-la en l'entorn de QtCreator i mira de carregar el Ferrari de la carpeta dataSet/Ferrari.obj. En aquesta primera execució, es llegeix el cotxe com si fos un únic objecte i es visualitza el cotxe centrat per que en el codi es fa manualment la transformació per a que es visualitzi.



2. Defineix i implementa primer els atributs i les constants d'una nova classe anomenada **Terra** (llegeix l'apartat 1.1 de l'enunciat). Cal cridar-lo des del menú de Fitxer: Crear Terra. Quan es crida aquest menú, sobre una finestra de diàleg on es demana l'amplada, la profunditat del terra i el pla de les Y's on situar el terra. Per això, implementa el mètode *newTerra*() de la classe GLWidget. Fes una primera prova per a veure'l, definint una dimensió inferior a 2. Si el defineixes més gran segurament no el veuràs sencer.

3. Modifica la classe **Escena** que guardarà tots el terra, els cotxes i els obstacles, que formen l'escenari del joc.

4. Crea les classes **Carrosseria** i **Roda** i modifica la classe **Cotxe** convenientment. Modifica el codi que permet la lectura de totes les parts del cotxe per fitxer. El punt origen de la capsa contenidora del cotxe s'ha de posicionar en les coordenades x_u, z_u, introduïdes per l'usuari i en la Y del terra (llegeix l'apartat 1.2 de l'enunciat).

Implementa el mètode calculCapsa3D a les classes objecte i cotxe.

5. Defineix i implementa la classe **Obstacle**. Quan es crida el menú Fitxer: New Obstacles es crearan a l'escena el numero de cubs, o obstacles, introduïts per l'usuari. Quan es crida aquest menú, s'obre una finestra de diàleg on es demana el numero d'obstacles a col·locar a l'escena. Per això, implementa el mètode *newObstacles*() de la classe GLWidget, que situa i escala aleatòriament els obstacles en el circuit. Per tal que es vegin els diferents obstacles que crea l'usuari, crea els cubs sobre el terra del circuit. Per això, per a cada nou cub que generis, copia les dades d'un cub prototipus i escala'l i trasllada'l.

6. Modifica el mètode *make*() de la classe objecte per a que tingui en compte l'escala, la posició i les rotacions demanades per l'usuari. Prèviament converteix l'objecte en una escala unitària.

7. Addicionalment, per tal que es visualitzi en el món creat en el glwidget.cpp, suposa que la teva escena està inclosa dins de la capsa de dimensió 50x50x50 i amb centre el punt (0,0,0). Per a visualitzar l'escena completa, implementa en la classe glwidget.cpp, el mètode:

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

que permeti traslladar i escalar convenientment els objectes, segons la mida del món 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.

8. Afegeix dinamisme a l'escena, incloent la interacció amb l'usuari. Amb les tecles del teclat es desitja dotar de moviment al cotxe. Per això:

- sobrecarrega els mètodes *KeyPressEvent()* i *KeyReleaseEvent()* per a realitzar la Transformació Geomètrica corresponent al moviment del cotxe. Amb la fletxa cap amunt, el cotxe avança segons la direcció d'avançada definida per l'usuari. Amb la fletxa cap avall, el cotxe dona marxa enrere. Amb les fletxes de dreta i esquerra el cotxe gira un certs gras cap a la dreta i l'esquerra respectivament.
- Totes les parts del cotxe es traslladen i giren segons les tecles. A més a més les rodes del cotxe giren sobre si mateixes.
- considera que la velocitat és constant. Només a cada interacció de la tecla es mou el cotxe.
- el cotxe es mou sempre en el pla del terra i no pot excedir els límits del circuit.

9. Afegeix realisme controlant les col·lisions del cotxe amb els obstacles, quan l'usuari mou el cotxe. Per a controlar aquestes col·lisions, considera que hi ha col·lisió quan la capsa del cotxe interseca amb alguna de les capses dels obstacles.

3 Extensions opcionals

Ext1: Per afegir més realisme al moviment, pots considerar que les tecles de moviment provoquen l'animació continuada del cotxe. 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 del cotxe i calcular el seu desplaçament a cada event del Timer.

Ext2: Dota d'acceleració al cotxe controlant el temps entre cada event de tecla. Aixi quan més "clics" es fan a la tecla d'avançar, per exemple, més ràpid va el cotxe.

Ext3: Inclou obstacles que provenen d'un fitxer .obj enlloc de cubs. Hauràs de canviar el control de col·lisions que has implementat en el punt 8?

Ext4: Llegeix un altre cotxe per a un segon jugador en el joc. Aquest segon cotxe es pot controlar des d'altres tecles del teclat.