

Mètodes virtuals que cal implementar:

- **initializeGL ()**
 - Codi d'inicialització d'OpenGL.
 - Qt la cridarà abans de la 1^a crida a `resizeGL`.
- **paintGL ()**
 - Codi per redibuixar l'escena.
 - Qt la cridarà cada cop que calgui el.repaint. El `swapBuffers()` és automàtic per defecte.
- **resizeGL ()**
 - Codi que cal fer quan es redimensiona la finestra.
 - Qt la cridarà quan es creï la finestra, i cada cop que es modifiqui la mida de la finestra.

GLSL: Operacions

- Inicialitzacions variables
 - **tipus bàsics:** `float b = 2.6;`
 - **vectorials:** `vec3 p(1.,1.5,2.);`
`p = vec3(3.,2.5,1.);`
 - **matrius:** per columnes `mat2 m=mat2(1.,2.,3.,4.);`
`m = [1., 3.]`
`[2., 4.]`
- Accés als elements de vectors
 - **swizzling:** `vec4 v;`
`v.xyzw; / v.rgba; // vec4`
`v.xy; / v.rg; // vec2`
`v.zyx; // canvia ordre!`
- Operacions vectors - multiplicació component a component

GLSL: Funcions predefinides

- Moltes funcions predefinides, especialment en àrees que poden interessar quan tractem geometria o volem dibuixar:
 - **trigonomètriques** `radians()`, `degrees()`, `sin()`, `cos()`, `tan()`, `asin()`, `acos()`, `atan()` (amb 1 o 2 paràmetres)
 - **numèriques** (poden operar sobre vectors component a comp.)
`pow()`, `log()`, `exp()`, `abs()`, `sign()`, `floor()`, `min()`, `max()`
 - **sobre vectors i punts**
`length()`, `distance()`, `dot()`, `cross()`, `normalize()`

Sessió 2.2: Objecte qualsevol i càmera ortogonal

4 hores

Un cop hem vist a classe la sessió 2.2 de laboratori, pots continuar amb la llista d'exercicis següents. T'aconseuem que et guardis el resultat de l'exercici de la sessió anterior.

2.2 Exercicis:

1. ► Fes un mètode que a partir de dos punts (punt-mínim i punt-màxim) d'una capsà contenidora d'una escena, calculi i guardi en atributs de la classe (o en variables de sortida del mètode) el centre de l'escena (centre d'aquesta capsà contenidora) i el radi de l'esfera que conté la capsà (que passarà a ser el radi de l'escena). Utilitza aquest mètode per calcular el centre i radi de l'esfera que conté l'escena que estem visualitzant, en aquest cas, els punts mínim i màxim de l'escena es poden posar a mà perquè són coneguts sabent l'escena que es visualitza (punts extrems del terra + alçada del HomerProves, que és 2).
2. ► Modifica els paràmetres de les crides a `lookAt` del mètode `viewTransform ()` i a `perspective` del mètode `projectTransform ()` per a què usin les dades de l'esfera contenidora de l'escena calculada en l'exercici anterior. Amb això aconseguim una càmera en tercera persona que ens permet veure tota l'escena sencera i ocupant el màxim del viewport.

Recorda el que s'ha explicat a teoria sobre el càlcul dels paràmetres d'una càmera en tercera persona i aplica-ho en aquest exercici amb les dades de l'escena que has calculat en l'exercici anterior.

3. ► Afegeix a l'exercici que mostra el terra i el HomerProves la implementació necessària per a que, quan l'usuari fa una redimensió de la finestra (*resize*), no hi hagi deformació de l'escena ni es retallí el que s'estava veient.

Recorda el que s'ha explicat a teoria sobre com han de variar els paràmetres de l'òptica de la càmera quan es varia la relació d'aspecte del viewport.

4. ► Ara volem visualitzar una instància del model **Patricio.obj** enlloc del Homer que teníem fins ara, però aquest model no està creat per a que els seus vèrtexs estiguin ubicats en el volum de visió que tenim definit (si el pintes no es veurà). Has d'afegeixir al teu codi el càlcul de la **capsa contenidora del model** (a partir dels seus vèrtexs) i pintar el Patricio amb la seva base centrada al punt (0,0,0) i escalat de manera que la seva alcàda sigui 4.

Modifica també el terra i fes que faci 5x5 de mida i estigui centrat a l'origen de coordenades (al punt (0,0,0)).

Tanmateix, l'escena que acabem de construir no hi cap al volum de visió que tenim, caldrà doncs que calculis també els paràmetres d'una càmera perspectiva que et permeti veure aquesta escena centrada, sencera, i ocupant el màxim del viewport (càmera en tercera persona per a aquesta nova escena).

Nota: Fixa't que si abans has fet bé la programació del càlcul dels paràmetres de càmera a partir de les dades de l'escena, ara només hauries de modificar els punts mínim i màxim de l'escena i tot s'hauria de calcular correctament.

5. ► Modifica el mètode `projectTransform()` per afegir la possibilitat de canviar el tipus d'òptica entre l'òptica perspectiva i l'ortogonal, i afegeix el codi necessari per poder implementar l'òptica ortogonal. Fes que els paràmetres d'aquesta òptica ortogonal també siguin els adients per a que l'esfera contenidora de l'escena estigui completament dins del volum de visió (com amb la perspectiva).

L'usuari ha de poder decidir entre l'òptica perspectiva i l'ortogonal mitjançant la tecla O. Inicialment l'òptica serà perspectiva i canviará cada cop que l'usuari premi la tecla O.

6. ► Fes que el redimensionament de la finestra (*resize*) no deformi ni retallí tampoc quan s'usa aquest tipus de càmera (ortogonal).

```
MyGLWidget::~MyGLWidget() { }

void MyGLWidget::modelTransform ()
{
    // Matriu de transformació de model
    glm::mat4 transform (1.0f);
    transform = glm::scale(transform, glm::vec3(m.escala*escala));
    transform = glm::rotate(transform,rotacio,glm::vec3(0.0,1.0,0.0));
    transform = glm::translate(transform,glm::vec3(0-m.centreBase.x, 0-m.centreBase.y, 0-m.centreBase.z));
    glUniformMatrix4fv(transLoc, 1, GL_FALSE, &transform[0][0]);
}

void MyGLWidget::modeloTrans(){
    glm::mat4 transform (1.0f);
    transform = glm::scale(transform, glm::vec3(escala));
    //transform = glm::scale(transform, glm::vec3(2.0,1.0,2.0));
    glUniformMatrix4fv(transLoc, 1, GL_FALSE, &transform[0][0]);
}

void MyGLWidget::initializeGL (){
    BL2GLWidget::initializeGL();
    glEnable (GL_DEPTH_TEST);
    projectTransform();
    viewTransform();
}
```

```
void MyGLWidget::resizeGL (int w, int h) {
    // Aquest codi és necessari únicament per a MACs amb pantalla retina.
#ifndef __APPLE__
    GLint vp[4];
    glGetIntegerv (GL_VIEWPORT, vp);
    ample = vp[2];
    alt = vp[3];
#else
    ample = w;
    alt = h;
    ra_window = float (w) / float (h);
    projectTransform();
#endif
}

void MyGLWidget::paintGL () {
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glViewport (0, 0, ample, alt);
    glBindVertexArray(VAO1);
    // Pintem l'escena
    modelTransform();
    glDrawArrays (GL_TRIANGLES, 0, m.model.faces ().size () * 3);

    // Desactivem el VAO
    glBindVertexArray(0);

    glBindVertexArray(VAO2);
    // Pintem l'escena
    modeloTrans();
    glDrawArrays (GL_TRIANGLES, 0, 6);

    // Desactivem el VAO
    glBindVertexArray(0);
}

void MyGLWidget::calcularCapsa_Patricio() {
    float minx, miny, minz, maxx, maxy, maxz;
    minx = maxx = m.model.vertices()[0];
    miny = maxy = m.model.vertices()[1];
    minz = maxz = m.model.vertices()[2];

    for (unsigned int i = 3; i < m.model.vertices().size(); i += 3) {
        minx = fmin(minx, m.model.vertices()[i+0]);
        maxx = fmax(maxx, m.model.vertices()[i+0]);

        miny = fmin(miny, m.model.vertices()[i+1]);
        maxy = fmax(maxy, m.model.vertices()[i+1]);

        minz = fmin(minz, m.model.vertices()[i+2]);
        maxz = fmax(maxz, m.model.vertices()[i+2]);
    }

    m.minim = glm::vec3(minx, miny, minz);
    m.maxim = glm::vec3(maxx, maxy, maxz);
    m.centre = glm::vec3((m.minim + m.maxim) / 2.f);
    m.centreBase = glm::vec3((minx + maxx)/2.f, miny, (minz + maxz)/2.f);
    m.amplada = maxx - minx;
    m.profunditat = maxz - minz;
    m.alcada = maxy - miny;
    m.escala = 4. / (maxy-miny);
}
```

```

void MyGLWidget::creaBuffers() {
    m_model.load("./models/Patricio.obj");
    calcularCapsa_Patricio();
    glGenVertexArrays(1, &VA01);
    glBindVertexArray(VAO1);
    // Creació del buffer amb les dades dels vèrtexs
    GLuint VBO1;
    glGenBuffers(1, &VBO1);
    glBindBuffer(GL_ARRAY_BUFFER, VBO1);
    glBufferData(GL_ARRAY_BUFFER, sizeof(GLfloat) * m_model.faces().size() * 3 * 3, m_model.VBO_vertices(), GL_STATIC_DRAW);
    glVertexAttribPointer(vertexLoc, 3, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(vertexLoc);

    // Creació del buffer amb les dades dels vèrtexs
    GLuint VBO2;
    glGenBuffers(1, &VBO2);
    glBindBuffer(GL_ARRAY_BUFFER, VBO2);
    glBufferData(GL_ARRAY_BUFFER, sizeof(GLfloat) * m_model.faces().size() * 3 * 3, m_model.VBO_matdiff(), GL_STATIC_DRAW);
    glVertexAttribPointer(colorLoc, 3, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(colorLoc);

    // Desactivem el VAO
    glBindVertexArray(0);

    glm::vec3 Vertices2[6];
    Vertices2[0] = glm::vec3(-2.5, 0.0, 2.5);
    Vertices2[1] = glm::vec3(-2.5, 0.0, -2.5);
    Vertices2[2] = glm::vec3(2.5, 0.0, 2.5);
    Vertices2[3] = glm::vec3(-2.5, 0.0, -2.5);
    Vertices2[4] = glm::vec3(2.5, 0.0, 2.5);
    Vertices2[5] = glm::vec3(2.5, 0.0, -2.5);

    glGenVertexArrays(1, &VA02);
    glBindVertexArray(VAO2);

    GLuint VBO3;
    glGenBuffers(1, &VBO3);
    glBindBuffer(GL_ARRAY_BUFFER, VBO3);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices2), Vertices2, GL_STATIC_DRAW);
    // Activem l'atribut que farem servir per vèrtex (només el 0 en aquest cas)
    glVertexAttribPointer(vertexLoc, 3, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(vertexLoc);

    glBindVertexArray(0);
}

void MyGLWidget::carregaShaders() {
    BL2GLWidget::carregaShaders(); // cridem primer al mètode de BL2GLWidget
    projLoc = glGetUniformLocation(program->programId(), "proj");
    viewLoc = glGetUniformLocation(program->programId(), "view");
}

void MyGLWidget::calcula(glm::vec3 punt_minim, glm::vec3 punt_maxim) {
    centre.x = (punt_maxim.x+punt_minim.x)/2;
    centre.y = (punt_maxim.y+punt_minim.y)/2;
    centre.z = (punt_maxim.z+punt_minim.z)/2;
    radi = (glm::distance(punt_maxim, punt_minim))/2;
}

void MyGLWidget::projectTransform() {
    calcula(glm::vec3(-2.5, 0.0, -2.5), glm::vec3(2.5, 4.0, 2.5));
    glm::mat4 Proj;
    znear = 2*radi+radi;
    zfar = 2*radi+radi;
    FOV = 2 * (glm::asin(radi/(2*radi)));
    left = -radi;
    right = radi;
    bottom = -radi;
    top = radi;

    if (prespectiva) {
        if (ra_window < 1) FOV = 2 *(glm::atan(glm::asin(radi/(2*radi)))/(ra_window));
        Proj = glm::perspective(FOV, ra_window, znear, zfar);
    }
    else { //resize ortogonal
        if (ra_window > 1) {
            left = -radi*ra_window;
            right = radi*ra_window;
        }
        else {
            bottom = -radi/ra_window;
            top = radi/ra_window;
        }
        Proj = glm::ortho(left, right, bottom, top, znear, zfar);
    }
    // glm::mat4 Proj = glm::perspective(float(M_PI)/2.0f, 1.0f, 0.4f, 3.0f);
    glUniformMatrix4fv(projLoc, 1, GL_FALSE, &Proj[0][0]);
}

void MyGLWidget::viewTransform() {
    VRP = centre;
    OBS = VRP + 2*radi*glm::vec3(0,0,1);
    UP = glm::vec3(0,1,0);
    glm::mat4 View = glm::lookAt(OBS, VRP, UP);
    // glm::mat4 View = glm::lookAt(glm::vec3(0,0,1), glm::vec3(0,0,0), glm::vec3(0,1,0));
    glUniformMatrix4fv(viewLoc, 1, GL_FALSE, &View[0][0]);
}

void MyGLWidget::keyPressEvent(QKeyEvent* event) {
    makeCurrent();
    switch (event->key()) {
        case Qt::Key_S: { // escalar a més gran
            escala += 0.05;
            break;
        }
        case Qt::Key_D: { // escalar a més petit
            escala -= 0.05;
            break;
        }
        case Qt::Key_R: {
            rotacio += M_PI/4;
            break;
        }
        case Qt::Key_O : {
            if(prespectiva) prespectiva = false;
            else prespectiva = true;
            projectTransform();
            break;
        }
        default: event->ignore(); break;
    }
    update();
}

```

Sessió 2.3: Euler i nova escena

4 hores

En aquesta tercera sessió del bloc 2, continua amb la llista d'exercicis següents. T'aconsellem que et guardis el resultat de l'exercici de la sessió anterior.

2.3 Exercicis:

- Modifica el mètode `viewTransform()` per a que faci el càlcul de la matriu `view` a partir de les transformacions mitjançant angles d'Euler. Inicialitza aquests paràmetres per veure exactament el que es veia a l'exercici 4 de la sessió del dia anterior, és a dir, mateixa posició relativa entre escena i càmera.
Recorda el que s'ha explicat a teoria sobre el càlcul de la View Matrix a partir dels paràmetres dels angles d'Euler (VRP, dist, Ψ , Θ i φ).
- Afegeix la possibilitat que l'usuari pugui interactuar amb el ratolí per modificar els angles que giren la càmera respecte els eixos Y i X (en les transformacions d'Euler). Quan l'usuari mou el ratolí en horitzontal d'esquerra a dreta la càmera s'ha de moure sobre l'esfera de visió en direcció a la dreta. Quan l'usuari mou el ratolí en vertical de baix a dalt la càmera s'ha de moure sobre l'esfera de visió en direcció cap a dalt (considerem l'esfera de visió com aquella esfera virtual centrada en VRP i de radi dist sobre la que es mou l'observador quan es modifiquen els angles Ψ i Θ d'Euler).
- Afegeix a la implementació del teu programa (del que tens fins ara) la possibilitat de fer zoom-in (amb la tecla Z) i zoom-out (amb la tecla X) de manera que es modifiqui l'angle (FOV) de la càmera perspectiva. *També ho podeu fer (opcionalment) per a la càmera ortogonal modificant la mida del window.*
- Ara, per fer una escena una mica més complexa, modifica el teu codi (pots fer una còpia i guardar el que tenies) de manera que la teva aplicació pinta una escena que té un terra centrat a l'origen de mida 5x5 (el que ja tens), amb 3 patricios que estan escalats de manera que la seva alçada és 1 i posicionats sobre el terra de manera que el primer té el centre de la base de la seva capsa contenidora al punt (2,0,2), el segon el té al punt (0,0,0) i el tercer el té al punt (-2,0,-2). El primer Patricio estarà mirant cap a les Z positives (sense cap rotació), el segon estarà mirant cap a les X positives i el tercer estarà mirant cap a les Z negatives.
Cal tenir definida una càmera perspectiva que vegi l'escena sencera, centrada, sense deformació i ocupant el màxim del viewport (fixa't que les mides de l'escena sencera són conegudes, per tant són coneguts els punts de la seva capsa contenidora).
- Afegeix a l'escena anterior un quart Patricio de la mateixa mida que els altres tres i amb la seva base a la posició (1.5, 0, 0) i mirant cap a Z+. Fes que aquest quart Patricio, utilitzant l'animació amb QTimer, giri constantment al voltant de l'eix Y de l'escena (farà voltes al voltant del Patricio que està al (0,0,0)).

```
void MyGLWidget::initializeGL () {
    BL2GLWidget::initializeGL();

    calcula(glm::vec3(-2.5,0.0,-2.5),glm::vec3(2.5,4.0,2.5));
    znear = 2*radi-radi;
    zfar = 2*radi+radi;
    FOV = 2 * (glm::asin(radi/(2*radi)));
    left = -radi;
    right = radi;
    bottom = -radi;
    top = radi;
    VRP = centre;
    OBS = VRP + 2*radi*glm::vec3(0,0,1);
    UP = glm::vec3(0,1,0);
    anglee = 0;
    psi = 0;
    theta = 0;
    phi = 0;
    ratoli.inici_accio = false;
    ratoli.invertir_moviment = false;
    ratoli.sensitibity = 200;
    ratoli.modo_freestyle = false;

    glEnable (GL_DEPTH_TEST);
    projectTransform();
    viewTransform();
}

void MyGLWidget::viewTransform () {
    glm::mat4 View(1.0f);
    View = glm::translate(View, glm::vec3(0., 0., -(2*radi)));
    View = glm::rotate(View, -phi, glm::vec3(0., 0., 1.));
    View = glm::rotate(View, theta, glm::vec3(1., 0., 0.));
    View = glm::rotate(View, -psi, glm::vec3(0., 1., 0.));
    View = glm::translate(View, glm::vec3(-VRP.x, -VRP.y, -VRP.z));
    glUniformMatrix4fv(viewLoc, 1, GL_FALSE, &View[0][0]);
}
```

```

void MyGLWidget::keyPressEvent(QKeyEvent* event) {
    makeCurrent();
    switch (event->key()) {
        case Qt::Key_S: { // escalar a més gran
            escala += 0.05;
            break;
        }
        case Qt::Key_D: { // escalar a més petit
            escala -= 0.05;
            break;
        }
        case Qt::Key_R: {
            rotacio += M_PI/4;
            break;
        }
        case Qt::Key_O : {
            if(prespectiva) prospectiva = false;
            else prospectiva = true;
            projectTransform();
            break;
        }

        case Qt::Key_Z : // zoom-in
            if (FOV > 0.1) FOV -= float(M_PI)/40.; // en radians //serien
            // Per ortogonal diferent
            if (not prospectiva) {
                // modifiquem el window
                left += 0.08;
                right -= 0.08;
                bottom += 0.08;
                top -= 0.08;
            }
            projectTransform();
            break;

        case Qt::Key_X : // zoom-out
            if (FOV < M_PI) FOV += float(M_PI)/40.; // en radians
            if (not prospectiva) {
                left -= 0.08;
                right += 0.08;
                bottom -= 0.08;
                top += 0.08;
            }
            projectTransform();
            break;
        default: event->ignore(); break;
    }
    update();
}

void MyGLWidget::mousePressEvent (QMouseEvent *event) {
    makeCurrent();
    if (event -> buttons() == Qt::LeftButton &&
        !(event -> modifiers() & (Qt::ShiftModifier | Qt::AltModifier | Qt::ControlModifier))) { // click esquerra només
        ratoli.inici_accio = true;

        ratoli.pos_inicial.x = event->x();
        ratoli.pos_inicial.y = event->y();
    }
    //#[EXTRA]: amb shift + click moviment invertit
    else if (event -> buttons() == Qt::LeftButton && (event -> modifiers() & Qt::ShiftModifier)) {
        ratoli.inici_accio = true;
        ratoli.invertir_moviment = true;

        ratoli.pos_inicial.x = event->x();
        ratoli.pos_inicial.y = event->y();
    }
    //[] amb ctrl + click modo freestyle
    else if (event -> buttons() == Qt::LeftButton && (event -> modifiers() & Qt::ControlModifier)) {
        ratoli.inici_accio = true;
        ratoli.modo_freestyle = true;
        ratoli.sensitiblity = 300;
    }
    update();
}

void MyGLWidget::mouseMoveEvent (QMouseEvent *event) {
    makeCurrent();
    if (ratoli.inici_accio) {
        if (!ratoli.invertir_moviment) {
            psi += (event->x() - ratoli.pos_inicial.x)/ratoli.sensitiblity; // dividir tot entre 200, és un suavitzador
            theta -= (event->y() - ratoli.pos_inicial.y)/ratoli.sensitiblity;

            if (not ratoli.modo_freestyle) {
                ratoli.pos_inicial.x = event->x(); // la posicio actual passa a ser la anterior
                ratoli.pos_inicial.y = event->y();
            }
        }
        else { //#[Optional] invertir moviment
            psi -= (event->x() - ratoli.pos_inicial.x)/ratoli.sensitiblity;
            theta += (event->y() - ratoli.pos_inicial.y)/ratoli.sensitiblity;

            if (not ratoli.modo_freestyle) {
                ratoli.pos_inicial.x = event->x();
                ratoli.pos_inicial.y = event->y();
            }
        }
    }
    update();
}

void MyGLWidget::mouseReleaseEvent (QMouseEvent *event) {
    makeCurrent();
    ratoli.inici_accio = false;
    ratoli.invertir_moviment = false; // []
    if (ratoli.modo_freestyle) { // []
        ratoli.pos_inicial.x = event->x();
        ratoli.pos_inicial.y = event->y();
    }
    ratoli.modo_freestyle = false; // []
    event->ignore();
    update();
}

```

Funcions OpenGL per a uniforms

Altres crides possibles:

```
glUniform{1|2|3|4}{f|i|ui} // nombre de paràmetres depenen de 1|2|3|4
```

1 – tipus float (f), int (i), unsigned int (ui), bool (f|i|ui)

2 – tipus vec2 (f), ivec2 (i), uvec2 (ui), bvec2 (f|i|ui)

3 – tipus vec3 (f), ivec3 (i), uvec3 (ui), bvec3 (f|i|ui)

4 – tipus vec4 (f), ivec4 (i), uvec4 (ui), bvec4 (f|i|ui)

```
glUniform{1|2|3|4}{f|i|ui}v (GLint loc, GLsizei count, const Type *value);
```

{1|2|3|4} i {f|i|ui} – igual que crida anterior

count – nombre d'elements de l'array value, 1: un sol valor; >=1 array de valors

```
glUniformMatrix{2|3|4|2x3|3x2|2x4|4x2|3x4|4x3}fv
```

```
(GLint loc, GLsizei count, GLboolean transpose, const GLfloat *value);
```

{2|3|4|2x3|3x2|2x4|4x2|3x4|4x3} – defineix les dimensions de la matriu

count – nombre de matrius de l'array value

transpose – si la matriu s'ha de transposar

Matrius de transformació

- Mètodes de transformacions geomètriques de la glm:

```
translate (glm::mat4 m_ant, glm::vec3 vec_trans);  
// retorna el producte de m_ant per una matriu que fa una  
// translació pel vector vec_trans
```

```
scale (glm::mat4 m_ant, glm::vec3 vec_scale);  
// retorna el producte de m_ant per una matriu que fa un  
// escalat en cada direcció segons els factors vec_scale
```

```
rotate (glm::mat4 m_ant, float angle, glm::vec3 vec_axe);  
// retorna el producte de m_ant per una matriu que fa una  
// rotació de angle radians al voltant de l'eix vec_axe
```

Fixeu-vos que aquestes crides multipliquen la matriu de transformació que generen per la dreta a la m_ant. Per tant la crida:

```
TG = glm::translate (TG, glm::vec3 (-0.5, 0.5, 0.0));
```

és equivalent a fer:

```
TG = TG * translate(-0.5, 0.5, 0.0); // en notació de teoria
```

Animacions (QTimer)

Per afegir animacions automàtiques a les nostres aplicacions:

- Usarem la classe QTimer de QT

```
#include <QTimer>
```

- Declarem un atribut de la nostra classe d'aquest tipus
`QTimer timer;`

- I definim un nou “slot” a la nostra classe
`public slots:`

```
void animar();
```

Per afegir animacions automàtiques a les nostres aplicacions:

- Cal afegir lligam en codi cpp

(només 1 cop –constructor o initializeGL)

```
connect (&timer, SIGNAL (timeout()), this, SLOT (animar ()));
```

- Decidim quan començar i quan acabar animació

```
timer.start (16); // s'activa cada 16 milisegons (60 cops per segon)
```

```
timer.stop (); // atura animació
```

- Exemple rutina per animació

```
void animar () {  
    makeCurrent ();  
    angleGir += increment; // exemple del que es vol animar  
    update ();  
}
```

Posició del focus de llum

Relativa a:

- L'escena – la posició del focus en SCA
 - Posició fixa del focus respecte a l'escena
 - Multiplicar posFocus per view Matrix per tenir-la en SCO
- La càmera – la posició del focus en SCO
 - Posició fixa respecte a la càmera
 - posFocus ja està en SCO directament
- Un model – la posició del focus en SCM
 - Posició fixa respecte al model d'un objecte
 - Multiplicar posFocus per (view * TG) igual que al model

3.1 Exercicis:

1. ► El càlcul d'il·luminació el farem sempre en coordenades d'observador, per fer això, caldrà que els paràmetres que se li passen a les funcions **Difus** i **Especular** estiguin tots en aquest sistema de coordenades. Això vol dir que en el Vertex Shader caldrà passar a coordenades d'observador la posició del vèrtex, la normal del vèrtex i la posició del focus de llum abans de cridar a les routines corresponents amb els paràmetres corresponents.
Fes aquests canvis de coordenades i calcula els paràmetres necessaris per fer el càlcul del color del vèrtex amb el model d'il·luminació de Lambert al Vertex Shader (recorda que Lambert inclou el component ambient i el difús).
2. ► Canvia el model d'il·luminació al de Phong (que inclou ambient, difús i specular) calculant de manera adient els paràmetres necessaris per a cada crida. Observes diferències en la il·luminació?
3. ► Ara modificarem els valors del material del terra+paret. Dóna-li els valors necessaris per a que siguin d'un material blau brillant (com si fos plàstic brillant).
4. ► Modifica la posició del focus de llum de manera que aquest focus quedí just a la part esquerra del Patricio, a la posició (1, 0, 1) en SCA (Sistema de Coordenades de l'Aplicació).
5. ► Converteix en uniforms tant la posició del focus de llum com el seu color. Fes que aquests uniforms es passin des de l'aplicació, és a dir des de la classe MyGLWidget.
6. ► Afegeix al codi de la classe MyGLWidget la possibilitat que amb les tecles K i L el focus de llum es mogui en la direcció de les X- i X+ respectivament. Podeu posar el focus de llum a la posició (1, 1, 1) en SCA i veure com passa per davant del Patricio.

```
MyGLWidget::~MyGLWidget() {  
}  
  
void MyGLWidget::initializeGL ()  
{  
    // Cal inicialitzar l'ús de les funcions d'OpenGL  
    initializeOpenGLFunctions();  
  
    glClearColor(0.5, 0.7, 1.0, 1.0); // defineix color de fons (d'esborrat)  
    glEnable(GL_DEPTH_TEST);  
    carregaShaders();  
    creaBuffersPatricio();  
    creaBuffersTerraIParet();  
  
    iniEscena();  
    iniCamera();  
}  
  
void MyGLWidget::iniMaterialTerra ()  
{  
    // Donem valors al material del terra  
    amb = glm::vec3(0.2,0,0.2);  
    diff = glm::vec3(0,0,0.8);  
    spec = glm::vec3(1,1,1);  
    shin = 100;  
}  
  
void MyGLWidget::mouseMoveEvent(QMouseEvent *e)  
{  
    makeCurrent();  
    // Aquí cal que es calculi i s'appliqui la rotació o el zoom com s'escaigui.  
    if (DoingInteractive == ROTATE)  
    {  
        // Fem la rotació  
        angleY += (e->x() - xClick) * M_PI / ample;  
        angleX += (e->y() - yClick) * M_PI / alt;  
        viewTransform ();  
    }  
  
    xClick = e->x();  
    yClick = e->y();  
  
    update ();  
}  
  
void MyGLWidget::keyPressEvent(QKeyEvent* event) {  
    makeCurrent();  
    switch (event->key()) {  
        case Qt::Key_K: // moure focus esquerra  
            rr = rr + glm::vec3(-0.2, 0, 0);  
            glUniform3fv(posFoc, 1, &rr[0]);  
            break;  
        case Qt::Key_L: // moure focus dreta  
            rr = rr + glm::vec3(0.2, 0, 0);  
            glUniform3fv(posFoc, 1, &rr[0]);  
            break;  
        default: BL3GLWidget::keyPressEvent(event); break;  
    }  
    update();  
}
```

```

#version 330 core

in vec3 fcolor;
out vec4 FragColor;

void main()
{
    FragColor = vec4(fcolor,1);
}

#version 330 core

in vec3 vertex;
in vec3 normal;

in vec3 matamb;
in vec3 matdiff;
in vec3 matspec;
in float matshin;

uniform mat4 proj;
uniform mat4 view;
uniform mat4 TG;

// Valors per als components que necessitem del focus de llum
uniform vec3 colorFocus;
vec3 llumAmbient = vec3(0.2, 0.2, 0.2);
uniform vec3 posFocus; // en SCA

out vec3 fcolor;

void main()
{
    fcolor = matdiff;
    vec3 vertSCO = (view * TG * vec4(vertex,1.0)).xyz;
    mat3 NormalMatrix = inverse (transpose (mat3 (view * TG)));
    vec3 NMNormal = NormalMatrix * normal;
    vec3 posF = (view * vec4(posFocus,1.0)).xyz;
    vec3 L = posF - vertSCO;
    L = normalize(L);
    NMNormal = normalize(NMNormal);
    vec3 Amb = Ambient();
    vec3 Dif = Difus(NMNormal,L,colorFocus);
    vec3 Phong = Especular(NMNormal,L,vec4(vertSCO,1.0),colorFocus);
    fcolor = Amb + Dif + Phong;
    gl_Position = proj * view * TG * vec4 (vertex, 1.0);
}

```

Executables de mostra:

Tecla 'O':

Canvia d'òptica

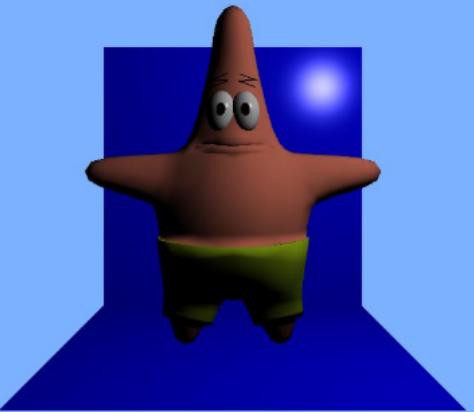
Tecla 'F':

Canvia de focus

de posició (1,1,1) en SCA
a focus de càmera.

Tecles 'K' i 'L':

Mouen focus sobre eix X



Sessió 3.2: Més il·luminació

4 hores

Un cop hem vist a classe la sessió 3.2 de laboratori, podeu continuar amb la llista d'exercicis següents. Us aconsellem, com sempre, que us guardeu el resultat de l'exercici de la sessió anterior.

3.2 Exercicis:

- Modifica el codi del Vertex Shader per a que el focus de llum sigui un focus de càmera, és a dir, que ja es donin les seves coordenades respecte el SCO (Sistema de Coordenades de l'Observador). Observa la diferència respecte al que tenies fent girar l'escena.
- Fes els canvis necessaris per a que el càcul de la il·luminació es faci al Fragment Shader. Caldrà que moguis les routines **Ambient**, **Difus** i **Especular** al Fragment Shader i també caldrà que facis arribar a aquest tots els paràmetres que necessites: Posició del vèrtex (pot estar ja en SCO), normal en el vèrtex (també pot estar ja transformada a SCO). La posició del focus de llum, com que és un uniform pot estar declarat directament en el Fragment Shader.
Quins canvis observes? Saps perquè?
- Implementa el canvi entre focus de càmera i focus d'escena. El focus d'escena ha d'estar a la posició (1,1,1) en SCA, i el de càmera exactament en la posició de la càmera. El canvi entre els dos tipus de focus el farem amb la tecla 'F'.

```

void MyGLWidget::carregaShaders() {
    // Creem els shaders per al fragment shader i el vertex shader
    QOpenGLShader fs (QOpenGLShader::Fragment, this);
    QOpenGLShader vs (QOpenGLShader::Vertex, this);
    // Carreguem el codi dels fitxers i els compilem
    fs.compileSourceFile("shaders/basicLlumShader.frag");
    vs.compileSourceFile("shaders/basicLlumShader.vert");
    // Creem el program
    program = new QOpenGLShaderProgram(this);
    // Li afegim els shaders corresponents
    program->addShader(&fs);
    program->addShader(&vs);
    // Linkem el program
    program->link();
    // Indiquem que aquest és el program que volem usar
    program->bind();

    // Obtenim identificador per a l'atribut "vertex" del vertex shader
    vertexLoc = glGetAttribLocation (program->programId(), "vertex");
    // Obtenim identificador per a l'atribut "normal" del vertex shader
    normalLoc = glGetAttribLocation (program->programId(), "normal");
    // Obtenim identificador per a l'atribut "matamb" del vertex shader
    matambLoc = glGetAttribLocation (program->programId(), "matamb");
    // Obtenim identificador per a l'atribut "matdiff" del vertex shader
    matdiffLoc = glGetAttribLocation (program->programId(), "matdiff");
    // Obtenim identificador per a l'atribut "matspec" del vertex shader
    matspecLoc = glGetAttribLocation (program->programId(), "matspec");
    // Obtenim identificador per a l'atribut "matshin" del vertex shader
    matshinLoc = glGetAttribLocation (program->programId(), "matshin");

    // Demanem identificadors per als uniforms del vertex shader
    transLoc = glGetUniformLocation (program->programId(), "TG");
    projLoc = glGetUniformLocation (program->programId(), "proj");
    viewLoc = glGetUniformLocation (program->programId(), "view");
    posFoc = glGetUniformLocation (program->programId(), "posFocus");
    if(camara == 0) rr = glm::vec4(0, 0, 0, 1);
    else rr = glm::vec4(1, 1, 1, 1);
    glUniform4fv(posFoc,1,&rr[0]);
    colorFoc = glGetUniformLocation (program->programId(), "colorFocus");
    glUniform3fv(colorFoc,1,&f[0]);
    camaras = glGetUniformLocation (program->programId(), "camera");
    glUniform1i(camaras,camara);
}

```

```

void MyGLWidget::keyPressEvent(QKeyEvent* event) {
    makeCurrent();
    switch (event->key()) {
        case Qt::Key_F:
            if(camara == 1) {camara = 0;rr = glm::vec4(0, 0, 0, 1);}
            else {camara = 1;rr = glm::vec4(1, 1, 1, 1);}
            glUniform1i(camaras,camara);
            glUniform4fv(posFoc,1,&rr[0]);

        break;
        case Qt::Key_K: // moure focus esquerra
            rr = rr + glm::vec4(-0.2, 0, 0,0);
            glUniform4fv(posFoc, 1, &rr[0]);

        break;

        case Qt::Key_L: // moure focus dreta
            rr = rr + glm::vec4(0.2, 0, 0,0);
            glUniform4fv(posFoc, 1, &rr[0]);

        break;
        default: BL3GLWidget::keyPressEvent(event); break;
    }
    update(); #version 330 core
}

out vec4 FragColor;
in vec3 matambFS;
in vec3 matdiffFS;
in vec3 matspecFS;
in float matshinFS;
in vec4 vertSCO;
in vec3 NMNormal;
in vec4 posFocusFS;

void main()
{
    vec3 NMNormal1;
    NMNormal1 = NMNormal;
    vec3 posF;
    posF = posFocusFS.xyz;
    vec3 L = posF - vertSCO.xyz;
    L = normalize(L);
    vec3 Amb = Ambient();
    vec3 Dif = Difus(NMNormal1,L,colorFocus);
    vec3 Phong = Especular(NMNormal1,L,vertSCO,colorFocus);
    FragColor = vec4(Amb + Dif + Phong,1);
}

```

```
#version 330 core
```

```
in vec3 vertex;
in vec3 normal;

in vec3 matamb;
in vec3 matdiff;
in vec3 matspec;
in float matshin;

out vec3 matambFS;
out vec3 matdiffFS;
out vec3 matspecFS;
out float matshinFS;

uniform mat4 proj;
uniform mat4 view;
uniform mat4 TG;
uniform vec4 posFocus; // en SCA
uniform int camera;

out vec4 vertSCO;
out vec3 NMNormal;
out vec4 posFocusFS;

void main()
{
    matambFS = matamb;
    matdiffFS = matdiff;
    matspecFS = matspec;
    matshinFS = matshin;
    vertSCO = (view * TG * vec4(vertex,1.0));
    mat3 NormalMatrix = inverse (transpose (mat3 (view * TG)));
    NMNormal = NormalMatrix * normal;
    NMNormal = normalize(NMNormal);
    if (camera == 1) { // El focus de Escena cal passar a SCO
        posFocusFS = view * posFocus;
    }
    else {posFocusFS = posFocus;}
    gl_Position = proj * vertSCO;
}
```

Sessió 3.3: Jugant amb focus i objectes

3 hores

La sessió 3.3 de laboratori es basa únicament en continuar fent exercicis que permeten assentar els coneixements adquirits sobre realisme. Podeu continuar amb la llista d'exercicis següents. Us aconsellem, com sempre, que us guardeu el resultat dels exercicis de la sessió anterior.

3.3 Exercicis:

- Implementa els canvis necessaris en el codi (tant del shader com del MyGLWidget) per a què en el càlcul del color es tingui en compte els dos focus de llum que tenim, el focus de càmera i el focus d'escena. Recorda que els components difús i specular s'han d'acumular per a cada focus que hi hagi.
- Modifica l'escena escalant el Patricio per a què faci alçada 0.3 i fes que el focus d'escena passi a estar a alçada 0.5 sobre el terra i posicionat just al damunt del Patricio (mateixes coordenades X i Z que el centre del Patricio). Fes que amb les tecles de les fletxes (Key_Left, Key_Right, Key_Up i Key_Down) el Patricio, juntament amb el focus de llum, es moguin per sobre del terra (modificant la seva posició en les coordenades X i Z).
- Fes que els dos focus de llum (el d'escena i el de càmera) es puguin encendre i apagar mitjançant les tecles '1' i '2' respectivament. Per "apagar" un focus de llum pots pensar en traure-li tota la seva intensitat de llum (color = (0,0,0)).
- Modifica el color del focus d'escena per a que sigui groc. De quin color es veu el terra il·luminat momés amb aquest focus? Perquè?
- Afegeix al teu codi la possibilitat que amb la tecla 'B' s'activi i desactivi el *Back-face culling*. Recorda que per a activar-lo només s'ha de fer un `glEnable (GL_CULL_FACE)`; i per a desactivar-lo un `glDisable (GL_CULL_FACE)`. Observa l'efecte que té en el pintat de la paret i el terra.

```

void MyGLWidget::keyPressEvent(QKeyEvent* event) {
    makeCurrent();
    switch (event->key()) {
        case Qt::Key_F:
            /*if(camara == 1) {camara = 0;rr = glm::vec4(0, 0, 0, 1);}*/
            else {camara = 1;rr = glm::vec4(1, 1, 1, 1);}
            glUniform1i(camaras,camara);
            glUniform4fv(posFoc,1,&rr[0]);*/
        break;
        case Qt::Key_Left: // moure focus esquerra
            rr = rr + glm::vec4(-0.2, 0, 0,0);
            glUniform4fv(posFoc, 1, &rr[0]);
            posPat.x -= 0.2;
            modelTransformPatricio();
        ;
        break;

        case Qt::Key_Right: // moure focus dreta
            rr = rr + glm::vec4(0.2, 0, 0,0);
            glUniform4fv(posFoc, 1, &rr[0]);
            posPat.x += 0.2;
            modelTransformPatricio();

        break;

        case Qt::Key_Up: // moure focus esquerra
            rr = rr + glm::vec4(0, 0, 0.2,0);
            glUniform4fv(posFoc, 1, &rr[0]);
            posPat.z += 0.2;
            modelTransformPatricio();

        break;

        case Qt::Key_Down: // moure focus dreta
            rr = rr + glm::vec4(0, 0, -0.2,0);
            glUniform4fv(posFoc, 1, &rr[0]);
            posPat.z -= 0.2;
            modelTransformPatricio();

        break;
        case Qt::Key_1:
            if (camara == 1) camara = 0;
            else camara = 1;
            glUniform1i(camaras,camara);
            break;

        case Qt::Key_2:
            if(escena == 1) escena = 0;
            else escena = 1;
            glUniform1i(escenes,escena);
            break;

        case Qt::Key_B:
            if(gl == 1) {gl = 0; glDisable(GL_CULL_FACE);}
            else {gl = 1; glEnable(GL_CULL_FACE);}
            break;

        default: BLGLWidget::keyPressEvent(event); break;
    }
    update();
}

#version 330 core
out vec4 FragColor;
in vec3 matambFS;
in vec3 matdiffFS;
in vec3 matspecFS;
in float matshinFS;

in vec4 vertSC0;
in vec3 NMNormal;
in vec4 posFocusFS;
in vec4 posFocusFS1;

uniform int cam;
uniform int esc;

// Valors per als components que necessitem del focus de llum
uniform vec3 colorFocus;
vec3 llumAmbient = vec3(0.2, 0.2, 0.2);

void main()
{
    vec3 NMNormal1;
    NMNormal1 = NMNormal;
    vec3 posF;
    posF = posFocusFS.xyz;
    vec3 posF1;
    posF1 = posFocusFS1.xyz;
    vec3 L = posF - vertSC0.xyz;
    L = normalize(L);
    vec3 L1 = posF1 - vertSC0.xyz;
    L1 = normalize(L1);
    vec3 Amb = Ambient();
    vec3 Dif;
    vec3 Phong;
    if (cam == 1 && esc == 1) {
        Dif = Difus(NMNormal1,L,colorFocus) + Difus(NMNormal1,L1,colorFocus);
        Phong = Especular(NMNormal1,L,vertSC0,colorFocus) + Especular(NMNormal1,L1,vertSC0,colorFocus);
        FragColor = vec4(Amb + Dif + Phong,1);
    }
    else if (cam == 1 && esc == 0) {
        Dif = Difus(NMNormal1,L1,colorFocus);
        Phong = Especular(NMNormal1,L1,vertSC0,colorFocus);
        FragColor = vec4(Amb + Dif + Phong,1);
    }
    else if (cam == 0 && esc == 1) {
        Dif = Difus(NMNormal1,L,colorFocus);
        Phong = Especular(NMNormal1,L,vertSC0,colorFocus);
        FragColor = vec4(Amb + Dif + Phong,1);
    }
    FragColor = vec4(Amb + Dif + Phong,1);
}

```

```
#version 330 core
```

```
in vec3 vertex;
in vec3 normal;

in vec3 matamb;
in vec3 matdiff;
in vec3 matspec;
in float matshin;

out vec3 matambFS;
out vec3 matdiffFS;
out vec3 matspecFS;
out float matshinFS;

uniform mat4 proj;
uniform mat4 view;
uniform mat4 TG;
uniform vec4 posFocus;
uniform vec4 posFocus1; // en SCA
uniform int camera;

out vec4 vertSCO;
out vec3 NMNormal;
out vec4 posFocusFS;
out vec4 posFocusFS1;

void main()
{
    matambFS = matamb;
    matdiffFS = matdiff;
    matspecFS = matspec;
    matshinFS = matshin;
    vertSCO = (view * TG * vec4(vertex, 1.0));
    mat3 NormalMatrix = inverse(transpose(mat3(view * TG)));
    NMNormal = NormalMatrix * normal;
    NMNormal = normalize(NMNormal);
    posFocusFS = view * posFocus1;
    posFocusFS1 = posFocus;
    gl_Position = proj * vertSCO;
}
```

Enunciat

L'objectiu de l'exercici és intentar fer una versió del clàssic joc "Snake" que es va popularitzar tant en 1998 perquè estava preinstal·lat als mòbils Nokia de l'època. Tot i mantenir una jugabilitat 2D voldrem visualitzar l'escena amb elements 3D: una serp que es mou sobre un terra i "creix" (s'allarga) al menjar uns items (bales en el nostre cas). A més a més, ficarem unes canonades (del Super Mario) per fer de frontera de l'espai de joc.

Primer haurem de pintar el terra amb les canonades fent de frontera o vorera, i després construirem i pintarem la serp considerant que té 3 parts: el cap, el cos i la cua (vegeu imatge del fitxer `escenaFinal.png`). Després haurem de fer que la serp es pugui moure, i que la part del cos sigui la que anirem replicant entre el cap i la cua, per fer la serp més llarga cada cop que es mengi una bola (vegeu imatge del fitxer `escenaFinal2.png`). Finalment, voldrem poder veure l'escena amb una càmera ortogonal en planta com si fos el joc 2D original (vegeu imatge del fitxer `escenaFinal3.png`). Et proporcionem un codi bàsic que crea i visualitza una escena formada per **un terra de 10x10** unitats ubicat sobre el pla XZ i centrat a l'origen, **una bola** escalada per a que tingui alçada 1.5 amb el centre de la seva base al punt (10, 0, 0) (`marblePos`), **una cua de serp** escalada per a que tingui alçada 1 amb el centre de la seva base al punt (4, 0, 0) (`tailPos`), **un cos de serp** escalat per a que tingui alçada 1 amb el centre de la seva base al punt (-1, 0, 0) (`bodyPos`), **un cap de serp** escalat per a que tingui alçada 1 amb el centre de la seva base al punt (-5, 0, 0) (`headPos`), i **una canonada** de decoració escalada per a que tingui alçada 1 amb el centre de la seva base al punt (-8, 0, 0) (vegeu imatge del fitxer `escenaInicial.png`). **Analitzeu el codi donat abans d'implementar funcionalitats.**

A partir d'aquest codi, resol els següents exercicis:

1. Modifica l'escena per a que:
 - (a) El terra faci **30x30** i segueixi centrat al punt **(0,0,0)**.
 - (b) Hi hagi **canonades** al voltant de tot el terra però a dins, no a fora, que facin **3** unitats d'alçada i es pintin amb una separació de 1 unitat. *Nota: Fixa't que t'hi cabran 30 canonades en cada costat del terra.*
 - (c) Es munti una serp sencera, d'alçada 1, mirant cap a les **X positives**, fent servir **el cap, la peça de cos i la cua** que ja tens, adientment col·locats i orientats (1 unitat de distància entre ells) amb les posicions dels centres de les seves respectives bases als punts **(0,0,0)**, **(-1,0,0)** i **(-2,0,0)**. Fixa't que tens les variables `headAngle` i `tailAngle` que cal que les utilitzis per fer l'orientació inicial.
 - (d) Hi hagi **1 bola** amb alçada **0.75** i amb el centre de la seva base al punt **(10,0,0)**.
2. Calcula els paràmetres d'una càmera perspectiva per tal de veure l'escena sencera, centrada i sense retallar. Per posicionar la càmera, has de fer servir els dos angles d'Euler (`psi`, `theta`) per tal de mostrar l'escena amb una inclinació **vertical inicial de 45 graus**. Afegeix també el codi d'interacció per al ratolí necessari per tal que es puguin modificar els angles d'Euler. Fixa't bé en el que ja tens implementat a la classe `LL2GLWidget`. Fes servir les variables `factorAngleX` i `factorAngleY` per traduir el desplaçament en pixels del ratolí a l'angle de rotació corresponent.
3. Volem que al premer les tecles `Key_Left`, `Key_Right`, `Key_Down` i `Key_Up` tota la serp es mogui 1 unitat en les respectives direccions `-X`, `+X`, `+Z`, `-Z`. L'orientació del cap ha de ser sempre mirant en la direcció de moviment, i la direcció de la cua ha de ser sempre mirant cap a la peça de cos a la qual ha d'anar enganxada.

Per resoldre aquest exercici, el codi proporcionat ja us dóna implementat un mètode `updateSnakeGame()` que actualitza la posició de la serp tenint en compte la direcció `direction` que estigui definida. Aquest mètode també s'encarrega de que quan el cap de la serp arriba a la posició de la bola se la menja, i llavors fa créixer la serp afegint una nova instància de cos de serp entre el cap i la cua. Un cop la serp ha menjat la bola, també tenim un mètode `computeRandomMarblePosition()` que fa aparèixer una nova bola a una posició aleatòria sobre el terra (en una posició vàlida que no està ocupada per la serp ni per les canonades).

El moviment de la serp també ha de ser restringit. Cal que **completis la implementació del mètode `checkPosition` (`glm::vec3 pos`)** que s'ha d'encarregar de comprovar si es pot fer el moviment següent o no. Per fer això cal controlar que el cap de la serp no pot anar a un espai que estigui ja ocupat pel seu propi cos (o cua) ni on hi hagi canonades (veure imatge del fitxer `escenaFinal2.png`).

4. Afegeix al codi una segona **càmera ortogonal en planta** que faci l'efecte de visualitzar el joc com si fos l'original en 2D (veure imatge del fitxer `escenaFinal3.png`). Fixa't que ha de coincidir el moviment de la serp provocat per les tecles amb l'orientació del tauler (Up - amunt, Down - avall, Left - esquerra i Right - dreta). Aquesta càmera s'activa/desactiva amb la tecla C.

En aquesta càmera no ha de funcionar la rotació dels angles d'Euler però sí que cal evitar retallats en fer un resize.

5. Afegeix el tractament de la tecla R de manera que permeti reinicialitzar l'escena i la càmera al resultat dels exercicis 1 i 2 (és a dir que es vegi tot com a la imatge del fitxer `escenaFinal1.png`).

6. **Opcional:** Afegeix una animació a l'aplicació de manera que quan l'usuari prem la tecla T s'activa un *timer* que fa que el moviment de la serp es fa de manera automàtica, fent que cada 200 mil·lisegons tota la serp avanci 1 unitat en la direcció en la que està mirant el cap. En premer la tecla T un altre cop es desactiva el *timer*. Nota: Fixa't que si el *timer* està activat les tecles no han de fer avançar la serp (perquè ja ho fa el *timer*) però sí que han de direccionar-la.

```
MyGLWidget::MyGLWidget(QWidget *parent=0) : LL2GLWidget(parent)
{
    connect(&timer, SIGNAL(timeout()), this, SLOT(moviment()));
}

void MyGLWidget::initializeGL ()
{
    // Cal inicialitzar l'ús de les funcions d'OpenGL
    initializeOpenGLFunctions();
    theta = M_PI/4;
    psi = 0.0;
    ortogonal = false;
    automatic = false;
    timer.stop();
    glEnable(GL_DEPTH_TEST);

    glClearColor(0.5, 0.7, 1.0, 1.0); // defineix color de fons (d'esborrat)
    carregaShaders();
    creaBuffersModels();
    creaBuffersTerra();
    iniEscena();
    iniCamera();
    setMouseTracking(true);
}

void MyGLWidget::iniEscena()
{
    centreEscena = glm::vec3(0,1.5,0);
    radiEscena = glm::distance(centreEscena,glm::vec3(15,3,15));

    headPos = glm::vec3(0,0,0);
    headAngle = 0;
    bodyPos.clear();
    bodyPos.push_back(glm::vec3(-1,0,0));
    tailPos = glm::vec3(-2,0,0);
    tailAngle = 0;

    marblePos = glm::vec3(10, 0, 0);
    direction = glm::vec3(1,0,0);
}
```

```
void MyGLWidget::iniCamera(){
    distancia = 2*radiEscena;
    obs = centreEscena + distancia* glm::vec3(0., 0., 1.);
    vrp = centreEscena;
    up = glm::vec3(0, 1, 0);
    fov = 2*(asin(float(radiEscena/(distancia))));
    znear = distancia-radiEscena;
    zfar = distancia+radiEscena;

    projectTransform();
    viewTransform();
}
```

```
#version 330 core
```

```

        ,
        case Qt::Key_S: {
            if (timer == 0) {
                timer = 1;
                t.start(50);
            }
            else {
                timer = 0;
                t.stop();
            }
            break;
        }
```

```
in vec3 vertex;
in vec3 normal;

in vec3 matamb;
in vec3 matdiff;
in vec3 matspec;
in float matshin;

uniform mat4 TG;
uniform mat4 Proj;
uniform mat4 View;
uniform int coche;
uniform vec4 TG_Coche;
uniform vec4 TG_Coche2;
uniform vec3 colorFoc;

vec3 verde = vec3(0.0, 1.0, 0.0);
vec3 rojo = vec3(1.0, 0.0, 0.0);
vec4 posFocus = vec4(0,0,0,1);

out vec4 vertSCO;
out vec3 NMNormal;
out vec4 posFocusFS;
out vec4 posFocusFSCoche;
out vec4 posFocusFSCoche2;
out vec3 fmatamb;
out vec3 fmatdiff;
out vec3 fmatspec;
out float fmatshin;
out vec3 colorFocus;
out vec3 colorFocusCoche;
```

```

#version 330 core

in vec3 fmatamb;
in vec3 fmatdiff;
in vec3 fmatspec;
in float fmatshin;
in vec4 vertSCO;
in vec3 NMNormal;
in vec4 posFocusFS;
in vec3 colorFocus;
const vec3 llumAmbient = vec3(0.1, 0.1, 0.1);
out vec4 FragColor;
in vec3 colorFocusCoche;
in vec4 posFocusFSCoche;
in vec4 posFocusFSCoche2;

void main()
{
    vec3 NMNormal1;
    NMNormal1 = NMNormal;
    vec3 posF;
    posF = posFocusFS.xyz;
    vec3 L = posF - vertSCO.xyz;
    L = normalize(L);
    vec3 posF1;
    posF1 = posFocusFSCoche.xyz;
    vec3 L1 = posF1 - vertSCO.xyz;
    L1 = normalize(L1);
    vec3 posF2;
    posF2 = posFocusFSCoche2.xyz;
    vec3 L2 = posF2 - vertSCO.xyz;
    L2 = normalize(L2);
    vec3 Amb = Ambient();
    vec3 Dif = Difus(NMNormal1,L,colorFocus) + Difus(NMNormal1,L1,colorFocusCoche) + Difus(NMNormal1,L2,colorFocusCoche);
    vec3 Phong = Especular(NMNormal1,L,vertSCO.xyz,colorFocus) + Especular(NMNormal1,L1,vertSCO.xyz,colorFocusCoche) + Especular(NMNormal1,L2,vertSCO.xyz,colorFocusCoche);
    FragColor = vec4(Amb + Dif + Phong,1);
}

```

```

void main()
{
    // Passem les dades al fragment shader
    fmatamb = matamb;
    if (coche == 1) {fmatdiff = matdiff*rojo; }
    else if (coche == 2) {fmatdiff = matdiff*verde; }
    else {fmatdiff = matdiff; }
    fmatspec = matspec;
    fmatshin = matshin;
    vertSCO = (View * TG * vec4(vertex,1.0));
    mat3 NormalMatrix = inverse (transpose (mat3 (View * TG)));
    NMNormal = NormalMatrix * normal;
    NMNormal = normalize(NMNormal);
    posFocusFS = posFocus;
    colorFocus = colorFoc;
    posFocusFSCoche = View * TG_Coche;
    posFocusFSCoche2 = View * TG_Coche2;
    colorFocusCoche = vec3(0.6,0.6,0.0);
    gl_Position = Proj * vertSCO;
}

```