Apunts Examen Lab IDI

Cameres Euler i Ortogonal

- [x] Normalment et proporcionen **una de les dues cameres** ja feta

- Crer un bool per a gestionar el canvi de camara

Euler

- Desde la funció `initializeGL()` es crida a `iniCamera()` on s'inicia la primera
- Per *aplicar* la camara hem de cridar a `projectTransform()` i `viewTransform()`
- Per poder moure la camara amb al ratolí hem de usar `mouseMoveEvent()`
- Al acabar el `mouseMoveEvent()` es cridara la `viewTransform()` per aplicar-ho

Inicialització

```
void ExamGLWidget::iniCamera ()
{
  angleY = 0.65; // Els angles incials de la camara, enunciat
  angleX = -1.2;
  camPlanta = false; // Bool creat per a gestionar l'us de dues cameres
  ra=float(width())/height(); // Calcul de la relació d'aspecte
  fov = float(M_PI/3.0); // Calcul del Field of View
  zn = radiEsc; // El zn i el zf quasi sempre son radiEsc i 3*radiEsc
  zf = 3*radiEsc;

  projectTransform (); // Crida a pT i vT per a "aplciar" els canvis
  viewTransform ();
}
```

ProjectTransform i ViewTransform

```
void ExamGLWidget::projectTransform ()
{
  Proj = glm::perspective(fov, ra, zn, zf); // Creades a iniCamera

  glUniformMatrix4fv (projLoc, 1, GL_FALSE, &Proj[0][0]); // Aplicar-ho
}
```

```
void ExamGLWidget::viewTransform ()
{
  View = glm::translate(glm::mat4(1.f), glm::vec3(0, 0, -2*radiEsc));
  View = glm::rotate(View, -angleX, glm::vec3(1, 0, 0));
  View = glm::rotate(View, angleY, glm::vec3(0, 1, 0));
  View = glm::translate(View, -centreEsc);

  glUniformMatrix4fv (viewLoc, 1, GL_FALSE, &View[0][0]); // Aplicar-ho
}
```

> En el `viewTransform()` com en qualsevol transformació els passos son:
>
> - Moure al centre la camara
> - Rotar-la en la inclinació adequada
> - Moure-la a la posició desitjada (normalment -2*radiEscena)

Moure la camara amb el ratolí

- [x] Aquesta funció es troba **normalment ja implementada** pels professors

```
void ExamGLWidget::mouseMoveEvent(QMouseEvent *e)
{
  makeCurrent();
  if ((DoingInteractive == ROTATE) && !camPlanta)
  {
    // Fem la rotació
    angleY += (e->x() - xClick) * M_PI / ample;
    angleX += (yClick - e->y()) * M_PI / alt;
    viewTransform ();
  }

  xClick = e->x();
  yClick = e->y();

  update ();
}
```

Ortogonal

- Modificar `viewTransform()` i pero `projectTransform()`

```
void MyGLWidget::viewTransform ()
{
    View = glm::lookAt(glm::vec3(0,2*radiEsc,0), glm::vec3(0,0,0),
        glm::vec3(1,0,0));
    glUniformMatrix4fv (viewLoc, 1, GL_FALSE, &View[0][0]);
}
```

```
void MyGLWidget::projectTransform ()
{
    Proj = glm::ortho(-radiEsc, +radiEsc, -radiEsc, +radiEsc, radiEsc,
        3*radiEsc);
    glUniformMatrix4fv (projLoc, 1, GL_FALSE, &Proj[0][0]);
}
```

Transformacions

```
TG = glm::translate(TG, glm::vec3(5,0,0)); // Moure a posicio final
TG = glm::rotate(TG, (float)angle, glm::vec3(0,1.0,0)); // Rotar
TG = glm::scale(TG, glm::vec3 (2.0*escala, 2.0*escala, 2.0*escala));
// Al fer la escala s'ha de mutiplicar per el "tamany" base
TG = glm::translate(TG, -centreBasePat); // Moure al centre

glUniformMatrix4fv (transLoc, 1, GL_FALSE, &TG[0][0]); // Aplicar-ho
```

Input Teclat

```
void MyGLWidget::keyPressEvent(QKeyEvent* event) {
  makeCurrent();
  switch (event->key()) {
  case Qt::Key_V: {
      // Input Code Here
    break;
    }
  case Qt::Key_1: {
      // Input Code Here
    break;
    }
    case Qt::Key_Right: {
      // Input Code Here
    break;
    }
    default: ExamGLWidget::keyPressEvent(event); break;
  }
  update();
}
```

Vertex Shader

```
void main()
{

    fmatamb = matamb;
    fmatdiff = matdiff;
    fmatspec = matspec;
    fmatshin = matshin;
    // Normalment fins aqui ja ve implementat

    mat3 normalMatrix = inverse(transpose(mat3(view * TG)));
    normalSCO = vec3(normalMatrix * normal);
    vertexSCO = view * TG * vec4(vertex, 1);

    gl_Position = proj * view * TG * vec4 (vertex, 1.0);
}
```

Fragment Shader

```
void main()
{
    vec3 NM = normalize(normalSCO); // normalSCO i vertexSCO del VertexSH
    vec3 L = normalize(posFocus - vertexSCO.xyz); //.xyz == vec3()
    // Normalment posFocus es un uniform que variem depenent del focus
    vec3 color = Especular(NM, L, vertexSCO, colFocus) +
    Difus(NM, L, colFocus) + Ambient(); // Sumem tots els focus
    FragColor = vec4(color,1); //Enviem la suma com a vec4()
}
```

> Les funcions Especular, Difus i Ambient **sempre** vindran implementades