FACULTAD DE INFORMÁTICA
Curso 2018-2019
Ejercicios Práctica 2

Parte 1: Cámara

- Añade a la clase Camera atributos eye, look, up, right, upward y front y los métodos:

  void setAxes(); // da valor a los tres ejes del sistema de la cámara
  void setVM() { viewMat = lookAt(eye, look, up); setAxes(); };

  Quita (comenta) los métodos pitch, yaw y roll, y adecua el código de los demás métodos.

- Añade a la clase Camera los métodos para desplazar la cámara en cada uno de sus ejes, sin cambiar la dirección de vista.

  void moveLR(GLdouble cs); // Left / Right
  void moveFB(GLdouble cs); // Forward / Backward
  void moveUD(GLdouble cs); // Up / Down

- Añade a la clase Camera atributos para el ángulo y el radio (p. ej. 1000) de la circunferencia que recorrerá la cámara con el método

  void orbit(GLdouble incAng, GLdouble incY); // modifica la posición de la cámara

  Tendrás que dar un valor adecuado al ángulo en los métodos set2D, set3D, …

- Añade a main dos nuevas variables globales, dvec2 mCoord para guardar las coordenadas del ratón y int mBot para guardar el botón pulsado. Añade también los callbacks para los eventos del ratón:

  glutMouseFunc(mouse);
  glutMotionFunc(motion);
  glutMouseWheelFunc(mouseWheel)

  void mouse(int button, int state, int x, int y) captura, en mCoord, las coordenadas del ratón (x, y) y en mBot el botón.

  void motion(int x, int y) captura las coordenadas del ratón, obtiene el desplazamiento con respecto a las anteriores coordenadas, y,
  Si el botón pulsado es el izquierdo, actualiza la posición de la cámara en la circunferencia.
  Si el botón pulsado es el derecho, desplaza la cámara en vertical y horizontal.

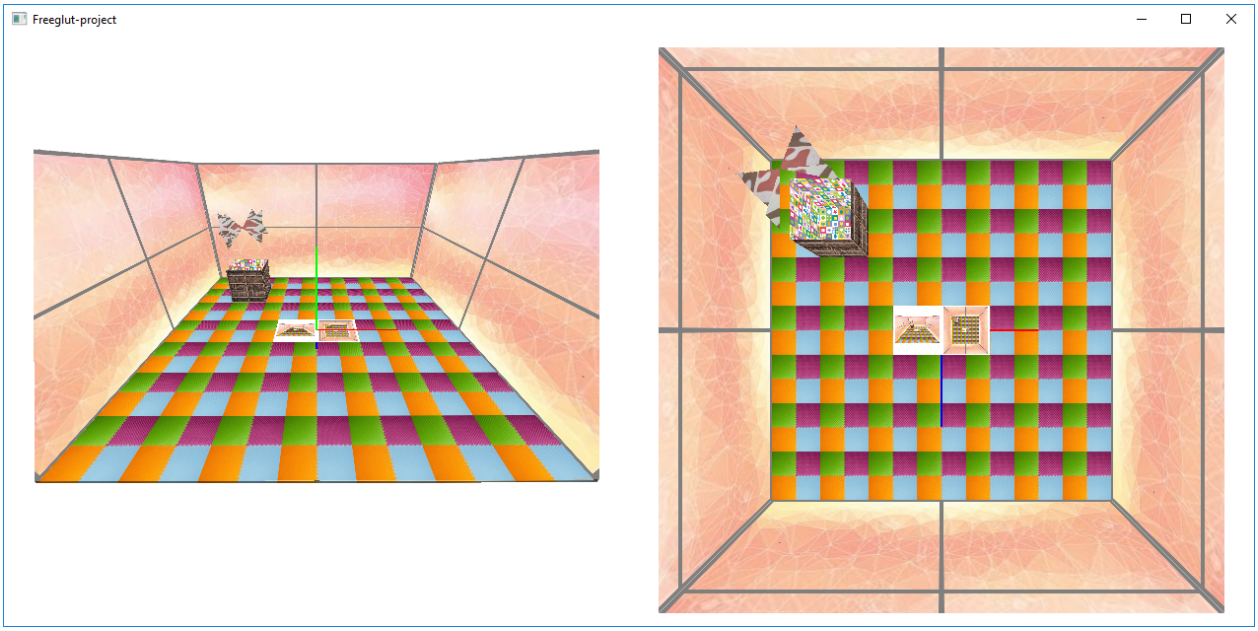
  void mouseWheel(int n, int d, int x, int y) Si no está pulsada ninguna tecla modificadora, desplaza la cámara en su dirección de vista. Si está pulsada la tecla CTRL escala la vista.

- Añade a la clase Camera un atributo bool orto; y un método changePrj() para cambiar de proyección ortogonal a perspectiva. Modifica los métodos de la clase Camera afectados por el cambio de proyección.

  Define la tecla 'p' para cambiar entre proyección ortogonal y perspectiva

- Opcional:

  Añade la opción de situar la cámara para realizar una vista cenital. Divide la ventana en dos puertos de vista y visualiza en el lado derecho la vista cenital.



Parte 2: Objetos cuádricos de GLU

Objetos cuádricos de GLU

- Objetos cuádricos de GLU

```
GLUquadricObj   * qObj;
GLUquadricObj   * esfera, *cilindro, *cono,  *cubo, *disco, *discP;
```

- Crear y destruir

```
qObj= gluNewQuadric();
gluDeleteQuadric(qObj);
```

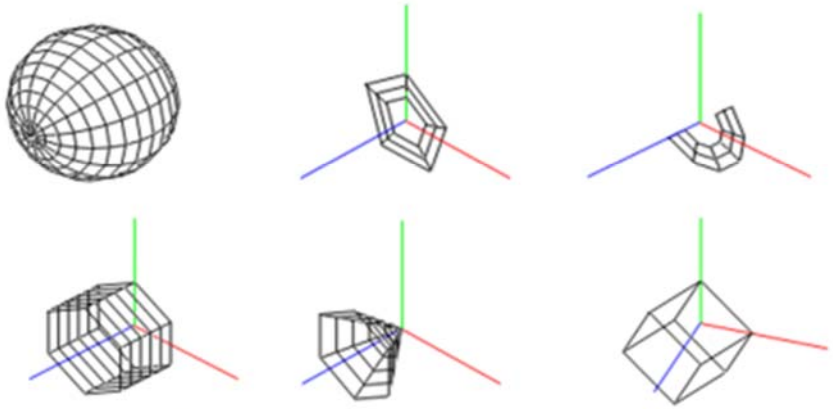
- Opciones de renderizado

```
gluQuadricDrawStyle(qObj, GLU_FILL);     // GLU_LINE, GLU_POINT
gluQuadricNormals(qObj, GLU_SMOOTH);  // GLU_FLAT
gluQuadricOrientation(qObj, GLU_OUTSIDE); // GLU_INSIDE
gluQuadricTexture(qObj, GL_FALSE);          // GLU_TRUE
```


Objetos cuádricos de GLU

Dibujar una esfera: gluSphere(qObj, radio, meridianos, paralelos);

Dibujar un disco: gluDisk(qObj, radioInterior, radioExterior, lados, anillos);



Dibujar un cilindro:

gluCylinder(qObj, radioDeLaBase, radioDeLaCima, altura, lados, paralelos);

- Iniciamos una nueva escena, deja el suelo y los ejes.

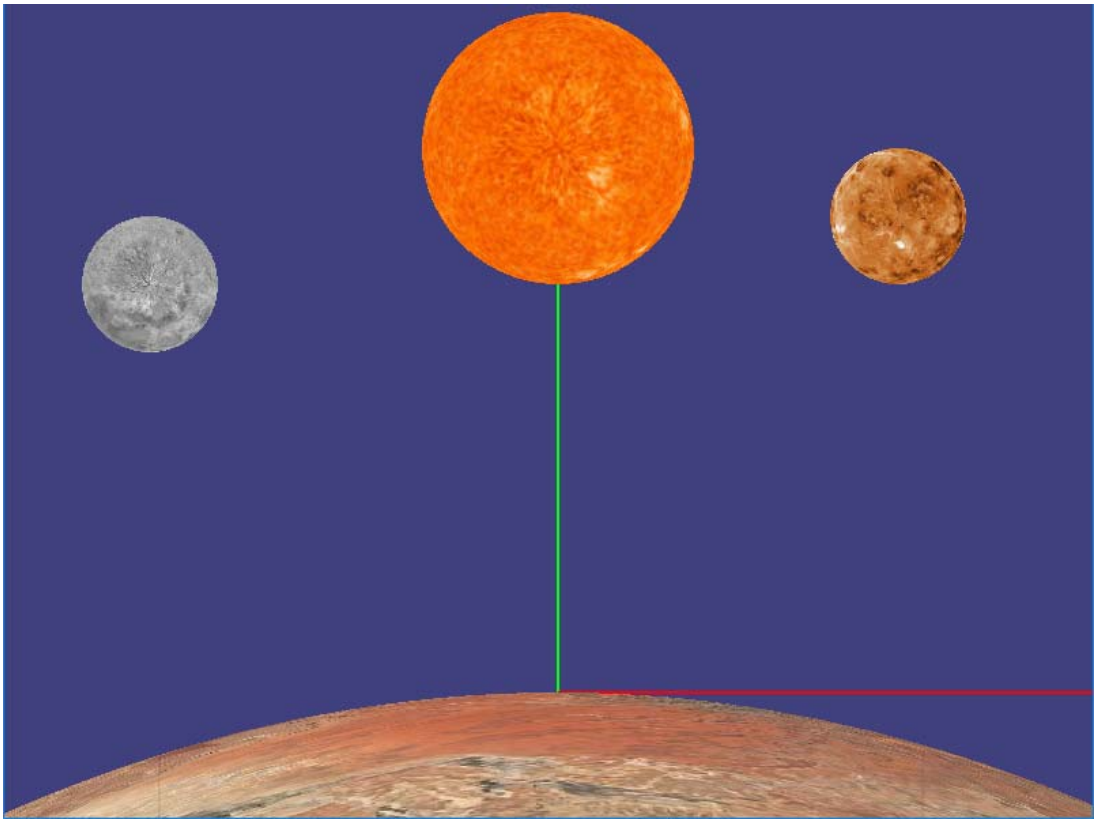
  No vamos a utilizar Blending.

  Para optimizar el renderizado de las nuevas entidades activa glEnable(GL_CULL_FACE).

- Añade la clase EntityMaterial que hereda de Entity. Añade un atributo para la textura Texture* texture; y un método setTexture(Texture* tex).

- Añade la clase Esfera que hereda de EntityMaterial. En lugar de generar una malla, vamos a utilizar un objeto cuádrico de la librería GLU. Define una constructora con un parámetro para el radio.

- Añade a la escena un array de punteros a texturas, y tres entidades de la clase Esfera, con distinto tamaño, distinta posición y distinta textura (puedes utilizar las imágenes venus, mars, moon, sun).



Luces y materiales (24 de abril)

- Añade al proyecto la clase Material, y las clases Light, DirLight y SpotLight.

  Consulta la tabla de materiales y define en la clase Material algunos métodos set para definir materiales de la tabla (por ejemplo setCooper(), …).

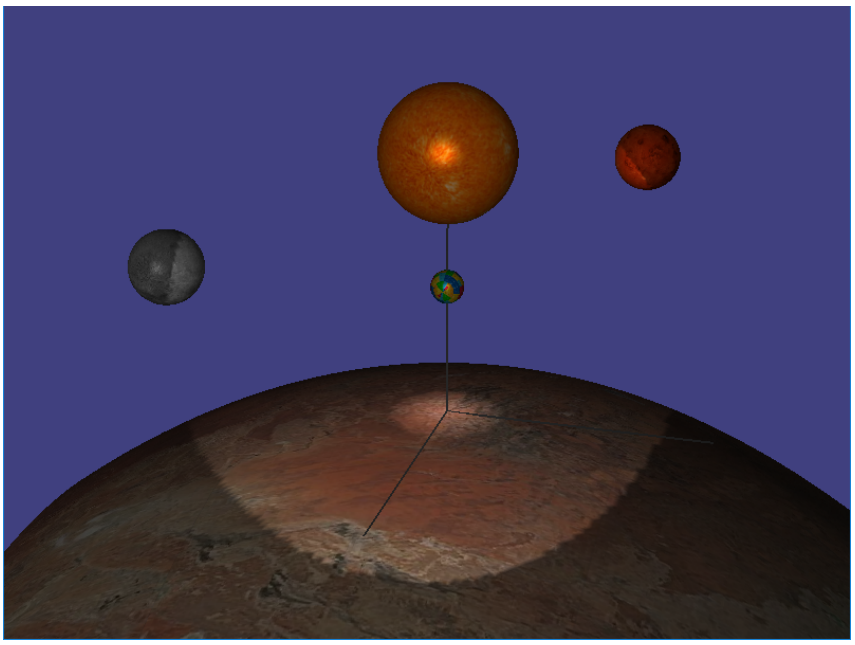
- Añade a la clase EntityMaterial un nuevo atributo para el Material y añade un método setMaterial(…). Al crear las esferas pon a cada una un material diferente.

- Modifica el método Esfera::render() para cargar el material antes de renderizar la malla (el objeto cuádrico). Recuerda que el método bind() de la clase Texture tiene un parámetro para indicar que se module el color de la textura y el obtenido por la iluminación.

- Añade a la escena dos atributos para dos luces: una direccional con vector de incidencia (0, -0.25, -1)), y la otra un foco en la cámara (posición y dirección de la cámara). Modifica init() para configurar las luces y render() para establecer las luces, antes de renderizar los objetos, con el método upload(…) de la clase Light. Recuerda activar la iluminación y la normalización de vectores.

- Define las teclas 'C' ('c') y 'V' ('v') para apagar (encender) el foco y la luz direccional respectivamente.

- Añade la clase EsferaLuz que hereda de Esfera con un nuevo atributo para un foco (puntero). El foco está en el centro de la esfera en dirección -Y. Añade a la escena una entidad de esta clase situada en el eje Y (utiliza la textura lego y un material con brillo especular). Define la tecla 'B' ('b') para apagar (encender) la esfera.



Parte 3: Mallas indexadas (10 de mayo)

- Añade a la clase Mesh un array para los vectores normales:

  dvec3* normals = nullptr;

  Modifica el método render() y la destructora para adecuarlos al nuevo atributo. Los comandos OpenGL para el array de normales:

  glEnableClientState(GL_NORMAL_ARRAY); glNormalPointer(GL_DOUBLE, 0, normals);

  glDisableClientState(GL_NORMAL_ARRAY);

- Añade la clase IndexMesh que hereda de Mesh con nuevos atributos para el array de índices: GLuint* indices = nullptr; GLuint numIndices = 0;

  Ahora el método render() tiene que utilizar el comando OpenGL glDrawElements(…) en lugar de glDrawArrays(…). Los comandos OpenGL para el array de índices:

  glEnableClientState(GL_INDEX_ARRAY); glIndexPointer(GL_UNSIGNED_INT, 0, indices);

  glDisableClientState(GL_INDEX_ARRAY);

- Añade la función static IndexMesh* generateGridTex(GLdouble lado, GLuint numDiv), para generar el array de vértices y el array de índices de una cuadrícula centrada en el plano Y=0, con la primitiva GL_TRIANGLES. Además, genera el array de coordenadas de textura para recubrir uniformemente con la imagen toda la cuadrícula.

  Añade la función static IndexMesh* generatePlanoCurvado(GLdouble lado, GLuint numDiv, GLdouble curvatura) que a partir de un Grid curva el plano modificando la coordenada Y de los vértices y añadiendo vectores normales, con las siguientes formulas:

  Dado un vértice del Grid V=(x, 0, z), la coordenada y se cambia a f(x, z) y su vector normal se define por n(x, z):

  f(x, z) = lado*curvatura/2 – curvatura/lado * (x*x) – curvatura/lado * (z*z)
  n(x, z) = (2*curvatura/lado * x, 1, 2*curvatura/lado * z)

- Añade la entidad Superficie con un plano curvado como malla. Utiliza como textura la imagen terreno, terrenoG, desierto o BarrenReds, y un material en grises para añadir a la textura el efecto de la iluminación.

Práctica 2 (17 de mayo))

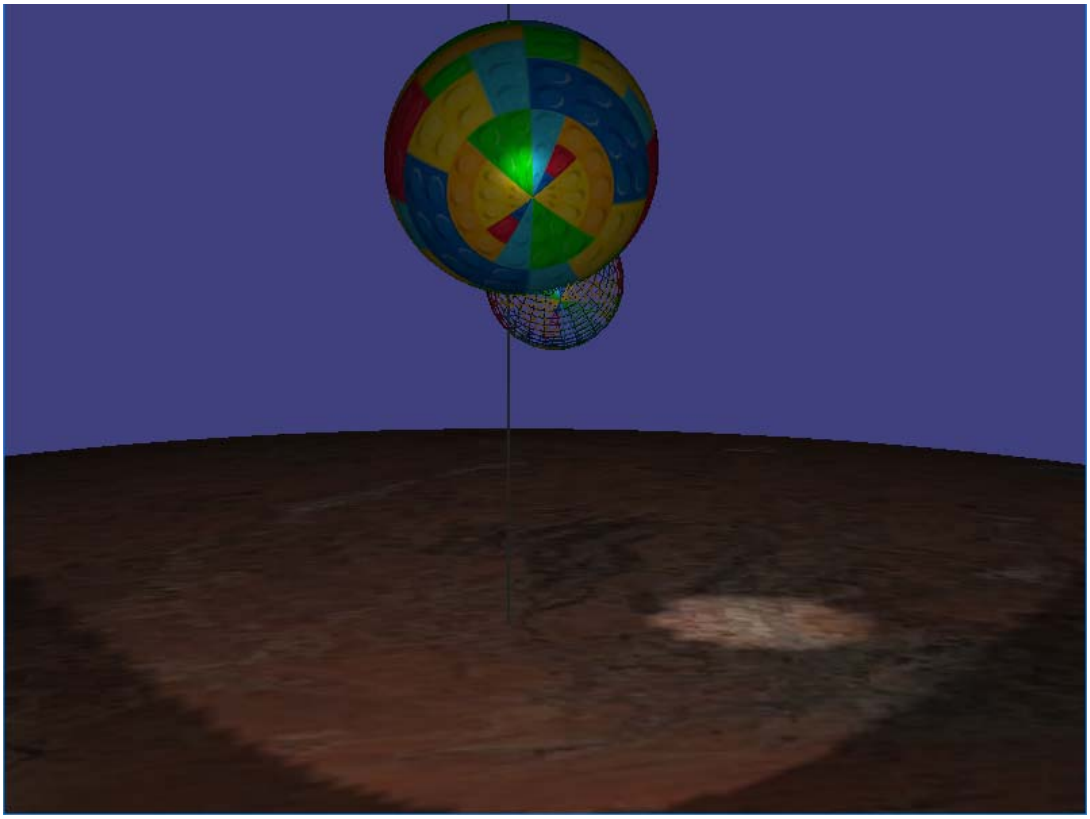
- Amplia la clase EsferaLuz para formar una entidad compuesta por la esfera con foco y otra esfera mayor según puedes ver en la imagen (la esfera con luz es la inferior y se renderiza en modo líneas). Utiliza la técnica del posicionamiento relativo y recuerda que el foco forma parte del objeto.

- Incorpora a la clase EsferaLuz una animación de forma que se desplace siguiendo una trayectoria dada por:

  T(ang) = (A cos(ang), B sin(ang) sin(ang), C sin(ang) cos(ang)),

  para 0 <= ang < 360, y A, B y C tres constantes (por ejemplo, B= altura del objeto, C = -A, y A = ½ lado de la superficie)

- Opcional: Añade a la esfera inferior un giro a derecha e izquierda de 90 grados (45 a cada lado).



Posicionamiento relativo

Si la entidad consta de varias partes (P0, P1, P2, …), siendo P0 la parte de referencia con matriz de modelado modelMat, añadimos como atributos matrices para el posicionamiento relativo de P1, P2,… con respecto a P0 (matP1, matP2, …), y a la hora de renderizar:

```
{
   dmat4 auxMat = modelMat;

   // modelMat = auxMat;
   uploadMvM(cam.getViewMat());     // envía a la GPU cam.getViewMat() * modelMat
   mesh->render();

   modelMat = auxMat * matP1;       // matP1 posiciona a P1 con respecto a P0
   uploadMvM(cam.getViewMat());     // envía a la GPU cam.getViewMat() * modelMat
   meshP1->render();   // meshP1 puede ser la misma mesh

   modelMat = auxMat * matP2;       // matP2 posiciona a P2 con respecto a P0
   uploadMvM(cam.getViewMat());     // envía a la GPU cam.getViewMat() * modelMat
   meshP2->render();   // meshP2 puede ser la misma mesh o meshP1
   …
   modelMat = auxMat;
 }
```

Las matrices matPi, posicionan al objeto Pi con respecto a P0 desde sus posiciones originales.

Si una parte es una luz, con matriz matPluz: luz->upload(cam.getViewMat() * matPluz)

Si la entidad tiene una animación, la matriz modelMat irá cambiando en el método update con la ecuación de la trayectoria, para lo que será necesario atributos como ángulo, radio, … que se irán actualizando en update.

Si una parte (Pa) a su vez tiene una animación propia, su matriz matPa irá cambiando también en el método update.