FACULTAD DE INFORMÁTICA
Curso 2018-2019
Ejercicios P1

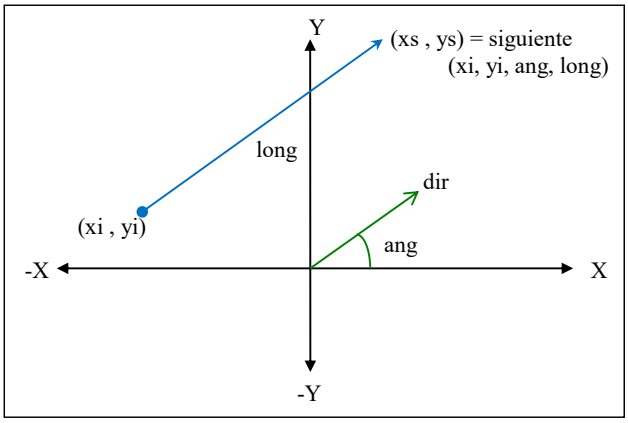
- Poliespirales (Dibujo de líneas)

Define la función static Mesh* generaPoliespiral(dvec2 verIni, GLdouble angIni, GLdouble incrAng, GLdouble ladoIni, GLdouble incrLado, GLuint numVert) que genera los vértices de la poliespiral que comienza en el vértice verIni y obtiene el siguiente vértice aplicando al anterior un desplazamiento en función del ángulo (dirección) y la longitud del lado:

siguiente(x, y, ang, long) = (x+long*cos(ang), y+long*sin(ang))

El ángulo y la longitud inicial son angIni y ladoIni, y se van incrementando en incrAng e incrLado respectivamente. Utiliza la primitiva GL_LINE_STRIP.

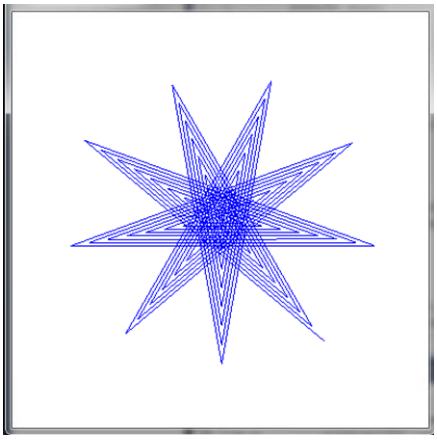
```
using namespace glm;
cos y sin para ángulos en radianes.
Para transforma grados a radianes: radians(degrees).
Por ejemplo: cos(radians(90))
```

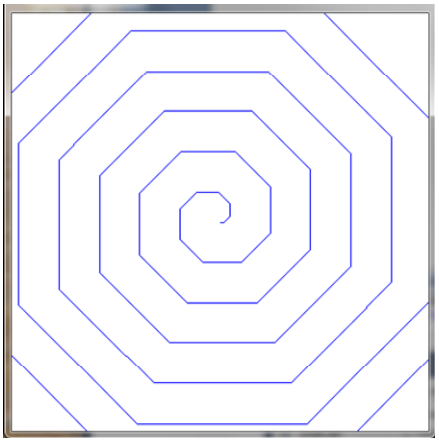


Define la clase Poliespiral heredando de Entity, y añade una entidad de esta clase a la escena. En el método render(…) establece el color de la poliespiral con glColor3d(…) y el grosor de las líneas con glLineWidth(2).

Prueba con los siguientes datos: verIni= (0, 0), angIni= 0

- incrAng= 160, lado= 1, incrLado= 1, numIter= 50
- incrAng= 72, lado= 30, incrLado= 0.001, numIter= 5
- incrAng= 60, lado= 0.5, incrLado= 0.5, numIter= 100
- incrAng= 89.5, lado= 0.5, incrLado= 0.5, numIter= 100
- incrAng= 45, lado= 1, incrLado= 1, numIter= 50





- Dragón (Dibujo de puntos)

  La generación de puntos basada en generar otro punto aplicando al anterior una transformación, se aplica a ciertas transformaciones dando lugar a figuras fractales.

  Para obtener el Dragón se utilizan dos transformaciones T1 y T2, eligiendo aleatoriamente una de ellas en cada iteración utilizando las probabilidades PR1 y PR2 respectivamente.

```
double azar= rand() / double(RAND_MAX);
if (azar < PR1) { … }  // T1
else { … }  // T2
```

  Define la función static Mesh* generaDragon(GLuint numVert) que genera los vértices del dragón (utiliza la primitiva GL_POINTS) que comienza en el (0, 0) y obtiene el siguiente vértice aplicando al anterior:

  - Con probabilidad PR1= 0.787473

    T1(x, y) = (0.824074 * x + 0.281482 * y - 0.882290,
    -0.212346 * x + 0.864198 * y - 0.110607

  - Con probabilidad PR2 = 1 - PR1 = 0.212527

    T2(x, y) = 0.088272 * x + 0.520988 * y + 0.785360,
    -0.463889 * x - 0.377778 * y + 8.095795

Define la clase Dragon heredando de Entity, y añade una entidad de esta clase a la escena. En el método render(…) establece el color del dragón con glColor3d(…) y el grosor de los puntos con glPointSize(2). Genera 3000 puntos y establece la matriz de modelado con una traslación de -40 en X y -170 en Y, y una escala de 40.

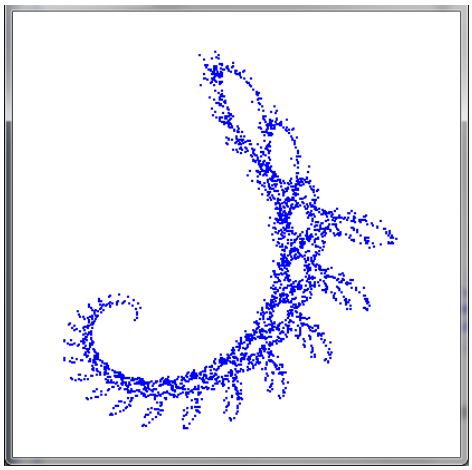
Utiliza las funciones de `glm` (`gtc/matrix_transform.hpp`) para transformaciones afines:

`translate(mat, dvec3(dx, dy ,dz))`: devuelve la matriz (`dmat4`) resultante de aplicar la traslación `(dx, dy, dz)` a la matriz `mat` (`dmat4`).

`scale (mat, dvec3(fs, fs, fs))`: devuelve la matriz (`dmat4`) resultante de aplicar la escala `(fs, fs, fs)` a la matriz `mat` (`dmat4`).

Por ejemplo:

```
modelMat = scale(modelMat, (5, 5, 5));
```



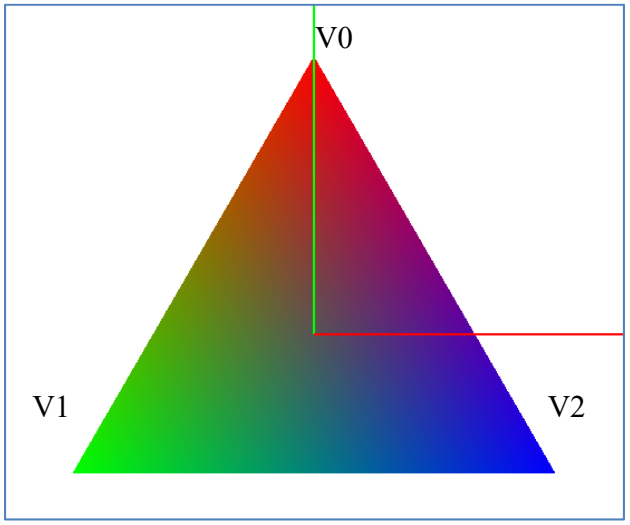
Prueba a cambiar el orden de las dos transformaciones

- TriánguloRGB

Define la función `static Mesh* generaTriangulo(GLdouble r)` que genera los tres vértices del triángulo equilátero de radio `r`, centrado en el plano Z=0 (utiliza la primitiva `GL_TRIANGLES`).

Un triangulo tiene dos caras (`BACK` y `FRONT`), y para identificarlas se usa el orden de los vértices en la malla. En OpenGL los vértices de la cara exterior (`FRONT`) se dan en orden contrario a las agujas del reloj (CCW).

En el ejemplo: V0, V1, V2



Utiliza la ecuación de la circunferencia, con centro C=(0, 0) y radio R=`r`:

$$x = Cx + R \cos(ang)$$
$$y = Cy + R \sin(ang)$$

En el ejemplo: angIni = 90
incrAng = 360/3

Define la función `static Mesh* generaTrianguloRGB(GLdouble r)` que añade al triángulo un color primario en cada vértice:

```
Mesh * generaTrianguloRGB(GLdouble r) {
      Mesh * m = generaTriangulo(r);
      …  // crear el array de colores
      return m;
}
```

Define la clase `TrianguloRGB` heredando de `Entity`, y añade una entidad de esta clase a la escena.

Podemos configurar el modo en que se rellenan los triángulos con el comando `glPolygonMode(…)`. Prueba, en el método `render(…)` de `TrianguloRGB`, este comando con distintas opciones y utiliza las flechas para cambiar la vista.

```
glPolygonMode(GL_BACK, GL_LINE)
glPolygonMode(GL_BACK, GL_POINT)

glPolygonMode(GL_FRONT_AND_BACK, GL_FILL)  // por defecto
```

- Rectángulo

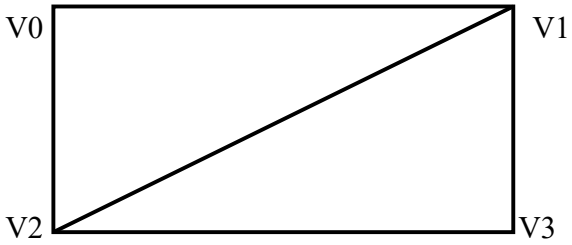
  Define la función `static Mesh* generaRectangulo(GLdouble w, GLdouble h)` que genera los cuatro vértices del rectángulo, centrado en el plano Z=0, de ancho `w`, y alto `h` (utiliza la primitiva `GL_TRIANGLE_STRIP`).

  Recuerda formar los triángulos en el orden contrario a las agujas del reloj.

  En el ejemplo: V0, V2, V1, V3

  Define los triángulos:

  V0, V2, V1 y V1, V2, V3

  Define la función `static Mesh* generaRectanguloRGB(GLdouble w, GLdouble h)` que añade un color a cada vértice.

  Define la clase `RectanguloRGB` heredando de `Entity`, y añade una entidad de esta clase a la escena.

  Gira el rectángulo 25º sobre el eje Z utilizando la transformación afín:

  ```
  rotate(mat, radians(ang), dvec3(eje de rotación))
  ```

  que devuelve la matriz (`dmat4`) resultante de aplicar la rotación a la matriz `mat` (`dmat4`).

  Prueba con el ángulo -25º.

- Escena 2D

  Compón una escena con todas las entidades anteriores utilizando las matrices de modelado para disponerlas en la escena.

  Posiciona algún gráfico de líneas o puntos delante del rectángulo con una traslación en el eje Z.

  Animación: Añade a la clase `Scene` un método `void update()` que indique a las entidades que se actualicen. Añade a la clase `Entity` el mismo método `update` con implementación vacía para que las subclases puedan redefinirlo. Define la tecla 'u' para indicar a la escena que se actualice.

  Modifica el método `render` en la clase `TrianguloAnimado` de forma que actualice su matriz de modelado para desplazarse describiendo una circunferencia a la vez que gira sobre su centro. Añade atributos para guardar los ángulos de giro. Redefine el método `update` para actualizar los ángulos.

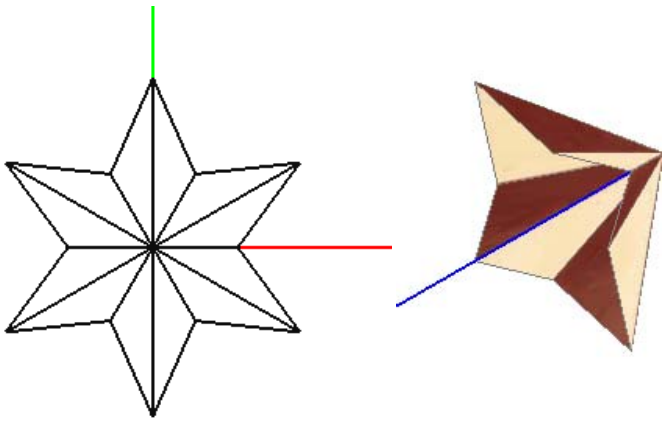
- Estrella 3D

Define la función static Mesh* generaEstrella3D(GLdouble re, GLdouble np, GLdouble h) que genera los vértices de una estrella de np puntas, centrada en el plano Z=h. Utiliza la primitiva GL_TRIANGLE_FAN con primer vértice V0 = (0, 0, 0).

El número de vértices es 2*np + 2.

Utiliza la ecuación de la circunferencia para generar los vértices. Puedes añadir un parámetro ri para el radio interior o utilizar re/2.

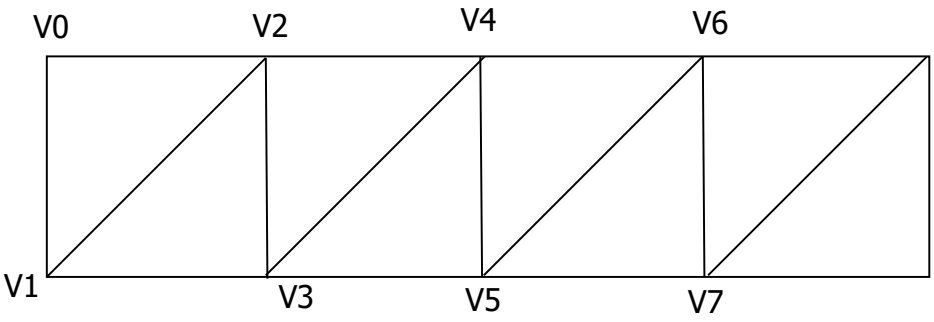
Recuerda generar los vértices en orden contrario a las agujas del reloj.

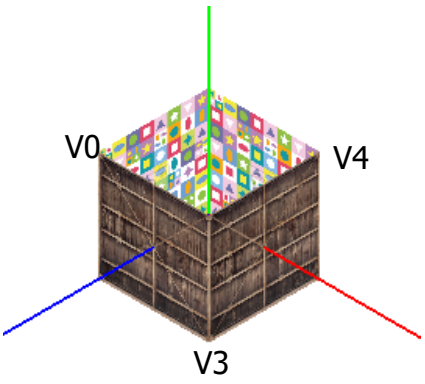


Define la clase Estrella3D heredando de `Entity`, y añade una entidad de esta clase a la escena (renderiza en modo líneas).

- Caja

Define la función static Mesh* generaContCubo(GLdouble l) que genera los vértices del contorno de una cubo, centrado en los tres ejes, de lado l. Utiliza la primitiva GL_TRIANGLE_STRIP.





El número de vértices es 10: 8 del cubo (V0,…, V7) + 2 para cerrar el contorno (V0, V1).

Define la clase Caja heredando de `Entity`, y añade una entidad de esta clase a la escena (renderiza en modo líneas).

- Escena 3D

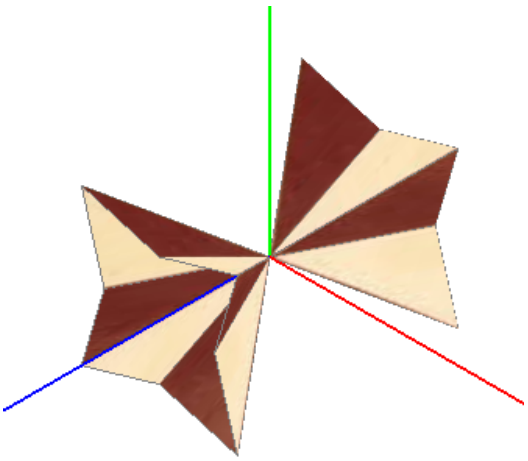
  Suelo (rectángulo centrado en el plano Y=0), caja sobre el suelo en el cuadrante –X, -Z, y estrella por encima de la caja.

- Suelo

  Entidad que renderiza un rectángulo centrado en el plano Y=0.

- Estrella 3D

  Modifica el método `render` de la clase Estrella3D para dibujar dos veces la estrella según aparece en la imagen. Puedes añadir un atributo para la matriz de modelado de la nueva estrella.



  Animación:

  Modifica el método `render` en la clase `Estrella3D` de forma que gire sobre su eje Z y sobre su eje Y. Añade atributos para guardar los ángulos de giro. Redefine el método `update` para actualizar los ángulos.

- Caja

  Añade a la clase Caja un rectángulo para el fondo.

Posicionamiento relativo

Si la entidad consta de varias partes (P0, P1, P2, …), siendo P0 la de referencia con matriz de modelado modelMat, a la hora de renderizar:

```
{
   dmat4 auxMat = modelMat;
   // si la entidad tiene animación y queremos que afecte a todas las partes ->
   // dmat4 auxMat = modelMat * matAnima;
   modelMat = auxMat;
   uploadMvM(cam.getViewMat()); // envía a la GPU cam.getViewMat() * modelMat
   meshP0->render();
   modelMat = auxMat * matP1;    // matP1 posiciona a P1 con respecto a P0
   uploadMvM(cam.getViewMat()); // envía a la GPU cam.getViewMat() * modelMat
   meshP1->render();
   modelMat = auxMat * matP2;    // matP2 posiciona a P2 con respecto a P0
   uploadMvM(cam.getViewMat()); // envía a la GPU cam.getViewMat() * modelMat
   meshP2->render();
   …
   modelMat = auxMat;
 }
```

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

- Animación (opcional)

Añade en main la función update() (sin argumentos) para el callback de glutIdleFunc. Está función será llamada cuando la aplicación esté desocupada y la utilizamos para actualizar los valores de animación. update() debe llamar a scene.update() cada cierto tiempo.

Añade una variable (GLuint last_update_tick) para capturar el último instante en que se realizó una actualización. Con glutGet(GLUT_ELAPSED_TIME) (devuelve los milisegundos transcurridos desde que se inició) podemos actualizar la variable y controlar el tiempo que debe transcurrir entre actualizaciones.
Añade también una variable bool para activar / desactivar la animación con la tecla 'U'.

Para ajustar las actualizaciones en función del tiempo transcurrido: Añade a la clase `Scene` un método `void update(GLuint timeElapsed)` que recibe el tiempo (en milisegundos) transcurrido desde la última actualización. Añade, a la clase `Entity`, el mismo método, con implementación vacía para que las subclases puedan redefinirlo. En main, update() debe llamar a scene.update(deltaTime) con el tiempo transcurrido desde la última actualización.

Texturas

Incorpora al proyecto el módulo cargador24BMP (cópialo en el directorio IGApp y añade los archivos al proyecto). Define la clase Texture y añade un atributo de tipo Texture a las entidades con textura. Carga la textura (con el método load) en la constructora y actívala (y desactivala) en el método render.

Modifica la clase Mesh para incorporar el array de coordenadas de textura (dvec2).

- Suelo con textura

  Define la función static Mesh* generaRectanguloTexCor(GLdouble w, GLdouble h, GLuint rw, GLuint rh) que añade coordenadas de textura para cubrir el rectángulo con una imagen que se repite rw veces a lo ancho y rh a lo alto.

  Modifica la constructora y el método render para renderizar el suelo con textura.

- Estrella con textura

  Define la función static Mesh* generaEstrellaTexCor(GLdouble r, GLdouble nL, GLdouble h) que añade coordenadas de textura a la estrella centrando la imagen en su vértice (0,0,0).

  Modifica la constructora y el método render para renderizar la estrella con textura.

- Caja con textura

  Define la función static Mesh* generaCajaTexCor(GLdouble *l*) que añade coordenadas de textura a la caja, repitiendo la imagen en cada lado.

  Modifica la constructora y el método render para renderizar la caja con dos texturas, una para el exterior de la caja y otra para el interior (incluido el fondo del cubo).

  Para renderizar solo el exterior o el interior, utiliza los comandos:

  ```
  glEnable(GL_CULL_FACE);
  glCullFace(GL_FRONT / GL_BACK);
  glDisable(GL_CULL_FACE);
  ```

- Foto

  Nueva entidad que renderiza un rectángulo sobre el suelo con una textura cuya imagen es la visualización de la escena en el renderizado anterior (front buffer). Para acceder a las dimensiones de la ventana: glutGet(GLUT_WINDOW_WIDTH / _HEIGHT)

  Añade a la clase Texture el método loadColorBuffer(…) con los comandos de OpenGL necesarios para copiar el color buffer en la textura: glCopyTexImage2D, glReadBuffer