FACULTAD DE INFORMÁTICA
Curso 2019-2020
Ejercicios P1

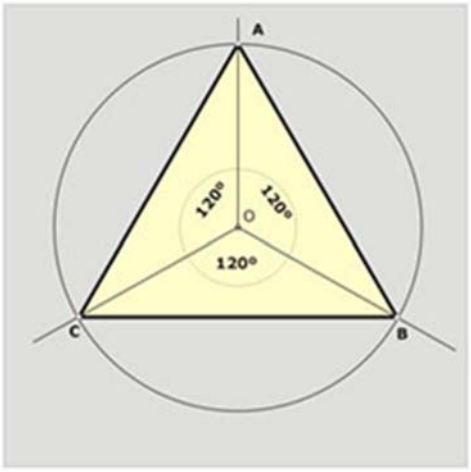
1. Polígono regular (Dibujo de líneas)

Define, en la clase Mesh, la función static Mesh* generaPoligono(GLuint numL, GLdouble rd) que genera los numL vértices del polígono regular inscrito en la circunferencia de radio rd centrada en el plano Z=0. Utiliza la primitiva GL_LINE_LOOP.

Para generar los vértices utiliza la ecuación de la circunferencia con centro C y radio R:

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

Para C=(0, 0) y R=rd.



Para el triángulo de la imagen generamos tres vértices, el primero con un ángulo inicial de 90º, y los siguientes incrementando el ángulo en 360º/numL (cuidado con la división). Recuerda pasar los grados a radianes:

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

Un triángulo tiene dos caras (FRONT y BACK), 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 la imagen: A, C, B.

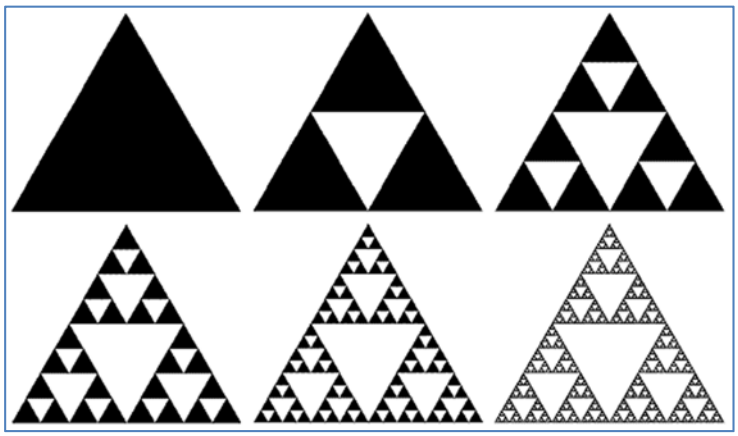
Añade a la clase Abs_Entity un atributo mColor para el color (dvec4), inícialo a 1 en la constructora (mColor(1)), y define un método para modificarlo. Define también la destructora.

Define la clase Poligono heredando de Abs_Entity, y redefine el método render(…) para establecer, antes de renderizar la malla, el color y el grosor de la líneas con glColor3d(r,g,b) y glLineWidth(2). Para acceder a las components del color: mColor.r, mColor.g, mColor.b. Después de renderizar la malla restablece, en OpenGL, los atributos a sus valores por defecto (1).

Añade a la escena (en el método init()) dos entidades de esta clase: un triangulo (amarillo) y una circunferencia (magenta).

Modifica el tamaño de la ventana, utiliza las teclas + y – para cambiar la escala, y las flechas para cambiar la vista.

2. Triangulo de Sierpinski (Dibujo de puntos)



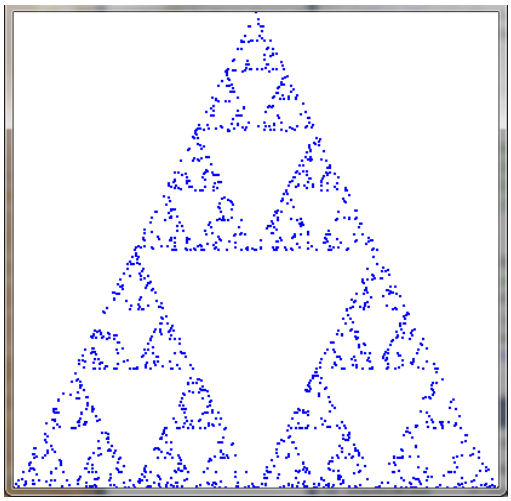
El matemático polaco Waclaw Sierpinski introdujo este fractal en 1919. El fractal de Sierpinski se puede construir tomando un triángulo cualquiera, eliminando el triángulo central que se obtiene uniendo los puntos medios de cada lado, y repitiendo hasta el infinito el mismo proceso en los tres triángulos restantes.

En la figura observamos hasta cinco iteraciones sucesivas, para el caso de un triángulo equilátero.

Otra forma de construir la figura del triángulo de Sierpinski es generando puntos a partir de los tres vértices T0, T1 y T2 de un triángulo equilátero inicial. Para ello:

- Se parte de un punto P0 elegido al azar de entre los tres vértices Ti.
- En cada iteración (k>0) se genera otro punto Pk calculando el punto medio entre el punto anterior (Pk-1) y uno de los tres vértices Ti elegido aleatoriamente.

Define la función `static Mesh* generaSierpinski(GLdouble rd, GLuint numP)` que genera la malla formada por numP vértices del triángulo equilátero de Sierpinski inscrito en la circunferencia de radio rd centrada en el origen. Utiliza la primitiva GL_POINTS.



Los tres vértices del triángulo inicial T0, T1 y T2, forman parte del triángulo de Sierpinski y serán los primeros vértices de la malla. Por tanto, para elegir aleatoriamente uno de ellos: vertices[rand()%3]

El punto medio Pm de dos puntos A=(Ax, Ay, Az) y B=(Bx, By, Bz) es la semisuma de las coordenadas de los puntos: Pm=((Ax + Bx) / 2, (Ay + By) / 2, (Az + Bz) / 2)

```
Mesh * generaSierpinski (…) {
    Mesh * triangulo = generaPoligono(3, rd);
    Mesh * mesh = new Mesh();
       … // crea la malla de Sierpinski
    delete triangulo; triangulo = nullptr;
    return mesh;
}
```

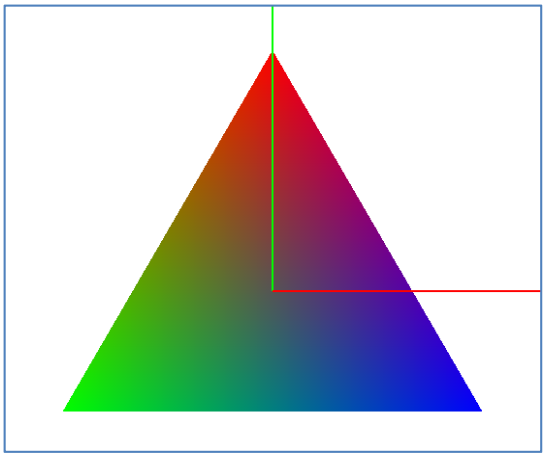
Define la clase Sierpinski heredando de Abs_Entity, redefine el método render(…) para establecer el grosor de los puntos con glPointSize(2) y el color con glColor4dv(value_ptr(mColor)). Recuerda restablecer, en OpenGL, los atributos a sus valores por defecto (1) después de renderizar la malla.

Añade a la escena una entidad de esta clase de color amarillo.

3. TrianguloRGB

Define la función static Mesh* generaTrianguloRGB(GLdouble rd) que añade al triángulo un color primario en cada vértice. Utiliza la primitiva GL_TRIANGLES.

```
Mesh * generaTrianguloRGB(GLdouble rd) {
        Mesh * mesh = generaPoligono(3, rd);
        … // añade el vector de colores
        return mesh;
}
```



Define la clase TrianguloRGB heredando de Abs_Entity, y añade una entidad de esta clase a la escena. Redefine el método render(…) para establecer que el triángulo se rellene por la cara FRONT y no por la cara BACK.

Podemos configurar el modo en que se rellenan los triángulos con el comando glPolygonMode(…):

```
glPolygonMode(GL_BACK, GL_LINE)
glPolygonMode(GL_BACK, GL_POINT)
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL)  // por defecto
```

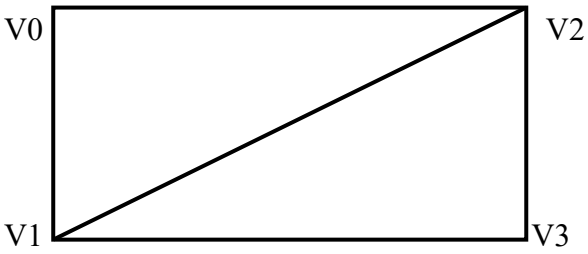
4. Rectángulo

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, V1, V2, V3

Define los triángulos: V0, V1, V2 y V2, V1, V3



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

Define la clase RectanguloRGB heredando de Abs_Entity, y añade una entidad de esta clase a la escena. Redefine el método render(…) para establecer que los triángulos se rellenen por la cara FRONT y no por la cara BACK.

5. Escena 2D

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

Utiliza las funciones de glm (tienes que incluir gtc/matrix_transform.hpp):

translate(mat, dvec3(dx, dy ,dz)): devuelve la matriz (dmat4) mat*translationMatrix, resultante de aplicar la translación (dx, dy, dz) a la matriz mat (dmat4).

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

rotate(mat, radians(ang), dvec3(eje de rotación)): devuelve la matriz (dmat4) mat*rotationMatrix, resultante de aplicar la rotación a la matriz mat (dmat4).

Por ejemplo: modelMat = scale(modelMat, (5, 5, 5));

Aplica al rectángulo una traslación en el eje Z de -100.
Aplica al triángulo RGB un giro en Z (prueba con 25º y -25º) y una translación en X e Y.

Animación:

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

Añade a la clase TrianguloRGB atributos para generar un desplazamiento del objeto describiendo una circunferencia a la vez que gira sobre su centro: los ángulos de giro y una matriz para la transformación. Redefine el método update para actualizar los ángulos y la matriz. Modifica el método render para aplicar las transformaciones posmultiplicando la matriz auxiliary aMat por la nueva matriz.