Informática Gráfica I

Curso 2023/2024

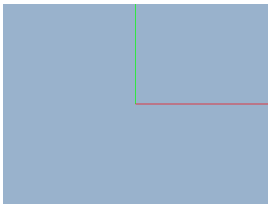
Facultad de Informática
Universidad Complutense de Madrid

Entrega I Apartados del 1 al 17

Fecha de entrega: 22 de febrero de 2024

Apartado 1

Localiza el comando que fija el color de fondo y cambia el color a $(0.6, 0.7, 0.8)$.



Apartado 2

En la clase `Mesh`, define el método:

```
static Mesh* generateRegularPolygon(GLuint num, GLdouble r)
```

que genere los `num` vértices que forman el polígono regular inscrito en la circunferencia de radio r, sobre el plano $Z = 0$, centrada en el origen. Utiliza la primitiva `GL_LINE_LOOP`. Recuerda que las ecuaciones de una circunferencia de centro $C = (C_x, C_y)$ y radio R sobre el plano $Z = 0$ son:

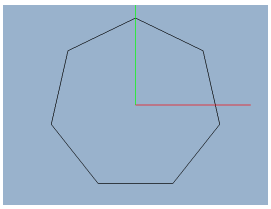
$$x = C_x + R \cdot \cos(\alpha)$$

$$y = C_y + R \cdot \cos(\alpha)$$

Genera los vértices empezando por el que se encuentra en el eje Y (α=90°) y, para los siguientes, aumenta el ángulo en `360°/num` (ojo con la división). Usa las funciones trigonométricas `cos(alpha)` y `sin(alpha)` de **glm**, que requieren que el ángulo `alpha` esté en radianes, para lo que puedes usar el conversor de **glm** para `radians(alpha)`, que pasa alfa grados a radianes.

Apartado 3

Define la clase `Polygon` que hereda de `Abs_Entity` y cuya malla se construye usando el método del apartado anterior. Incorpora un objeto de esta nueva clase a la escena. En la captura adjunta se muestra un heptágono regular.



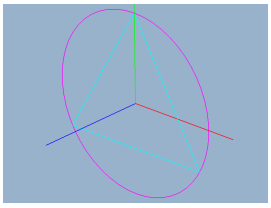
Apartado 4

Añade a la clase `Abs_Entity` un atributo **glm**`::dvec4 mColor`, para dotar de color a una entidad, sin tener que dar color a los vértices de su malla. Inicializa este atributo a 1 en la constructora (`mColor(1)`), y define sus métodos `get` y `set`. Modifica el método `render()` de la clase `Poligono` para que tenga en cuenta el color. Para establecer el color utiliza el comando `glColor4d(mColor.r, mColor.g, mColor.b, mColor.a)`.

No olvides restablecer el color por defecto antes de terminar la renderización.

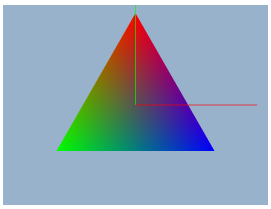
Apartado 5

Añade a la escena un triángulo cian y una circunferencia magenta como objetos de la clase `Polygon`, tal como se muestra en la figura.



Apartado 6

Define la clase `RGBTriangle` que hereda de `Abs_Entity` y cuyos objetos se renderizan como el de la captura de la imagen. Observa que solo tienes que añadir colores apropiados a los vértices de una malla triangular de la clase `Polygon`. Añade uno de estos triángulos a la escena.



Apartado 7

Redefine el método `render()` para establecer que el triángulo se rellene por la cara **FRONT** mientras que por la cara **BACK** se dibuja con líneas. Haz lo mismo, pero que las caras traseras se dibujen con puntos.

Apartado 8

Define el método:

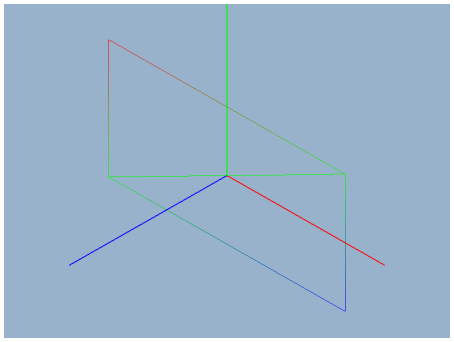
```
static Mesh* generateRectangle(GLdouble w, GLdouble h)
```

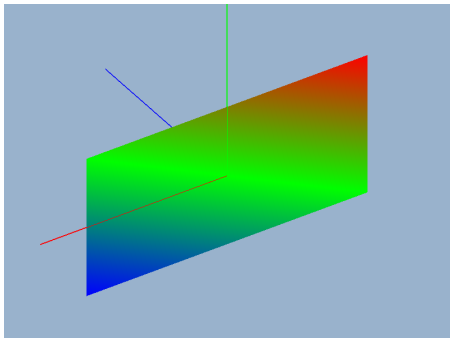
que genera los cuatro vértices del rectángulo centrado en el origen, sobre el plano $Z = 0$, de ancho w y alto h. Utiliza la primitiva `GL_TRIANGLE_STRIP`.

Define el método:

```
static Mesh* generateRGBRectangle(GLdouble w, GLdouble h)
```

que añade un color primario a cada vértice (un color se repite), como se muestra en las capturas. Define la clase `RGBRectangle` que hereda de `Abs_Entity`, y añade una entidad de esta clase a la escena. Redefine su método `render()` para establecer que los triángulos se rellenen por la cara **BACK** y se muestren con líneas, por la cara **FRONT**.



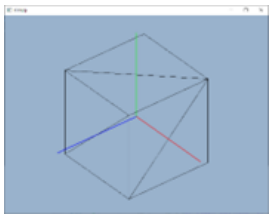


Apartado 9

Define el método:

```
static Mesh* generateCube(GLdouble longitud)
```

que construye la malla de un cubo (hexaedro) con arista de tamaño longitud, centrado en el origen. Define la clase Cubo que hereda de `Abs_Entity`, y añade una entidad de esta clase a la escena. Renderízalo con las caras frontales en modo línea (con color negro) y las traseras, en modo punto, como en la captura adjunta.

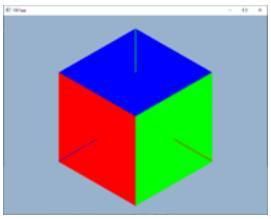


Apartado 10

Extiende la malla anterior con color en los vértices definiendo el método estático:

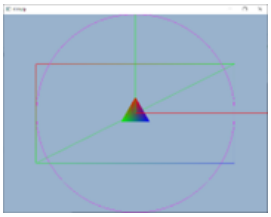
```
static Mesh* generateRGBCubeTriangles(longitud)
```

El color es el que se muestra en la captura. Define la clase `RGBCube` que hereda de `Abs_Entity`, y añade una entidad de esta clase a la escena.



Apartado 11

Construye sendas escenas, una bidimensional con un rectángulo como el del apartado 8 que contiene en su interior un pequeño triángulo RGB, como el del 6, al que rodea una circunferencia como la del apartado 3, y otra escena tridimensional con un cubo como el del apartado anterior.



Apartado 12

Define el método setter `setScene(id)` del atributo `mId` de la clase `Scene`. Implementa los eventos de teclado **'0'** y **'1'** para permitir cambiar entre la escena **0**, que será la bidimensional, y la **1**, que será la tridimensional.

Apartado 13

Añade a la clase `Abs_Entity` un método `virtual void update() {}` que se usa para modificar la `mModelMat` de aquellas entidades que la cambien, por ejemplo, en animaciones. Añade a la clase `Scene` un método `void update()` que haga que las entidades de `gObjects` se actualicen mediante su método `update()`. Define en `IG1App` el evento de la tecla **'u'** para hacer que la escena se actualice con una llamada a su método `update()`.

Apartado 14

Coloca el triángulo RGB de la escena **0** en el punto **(R, 0)**, siendo R el radio de la circunferencia de esa escena.

Apartado 15

Define el método `update()` en la clase `RGBTriangle` de forma que el triángulo de esta clase de la escena **0**, rote en horario sobre sí mismo a la par que lo hace en anti horario sobre la circunferencia.

Apartado 16

Programa el método `update()` (sin argumentos) en la clase `IG1App`, que es usado por el callback de `glutIdleFunc`. Haz que este método se ejecute en el evento de teclado **'U'**.

Apartado 17

(Opcional) Programa el método `update()` de la clase `RGBCube` tal como se muestra en la grabación "*update de la escena 1*".