Informática Gráfica I

Curso 2024/2025

Facultad de Informática
Universidad Complutense de Madrid

Entrega I Apartados del 1 al 17
Fecha de entrega: 13 de febrero de 2025

Instrucciones de la práctica

Para llevar a cabo este trabajo se tendrá en cuenta lo siguiente:

- Esta práctica se compone de varias escenas.
- Para realizar cada escena es necesario implementar una serie de pasos, en forma de apartados.
- El código que se desarrolle para cada escena deberá ser reutilizable por las escenas posteriores.
- Se tendrá especial atención a la calidad del código: inexistencia de fugas de memoria, código bien organizado, comentado y legible, se siguen los principios de la programación orientada a objetos, entre otros.

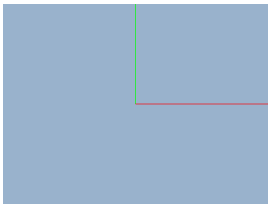
Rúbrica de evaluación

Criterio 1	**Pesos**
Escena 1	2.5
Escena 2	2.5
Escena 3	2.5
Calidad del código	1.5
Ausencia de fugas	1
Total	**10.0**
Opcional	+1
Defensa	-1.5

Escena 1: Polígonos sin relleno

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:

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

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

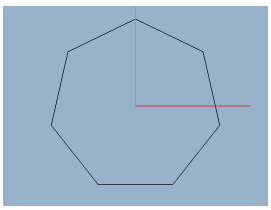
Genera los vértices empezando por el que se encuentra en el eje Y (α=90°) y, para los siguientes, aumenta el ángulo en `360°/num` (ojo con la división). Usa las funciones trigonométricas `cos(alpha)` y `sin(alpha)` de **glm**, que requieren que el ángulo `alpha` esté en radianes, para lo que puedes usar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

Apartado 3

Define una clase `SingleColorEntity` que extienda `Abs_Entity` con un atributo `mColor` de tipo **glm**`::vec4` para dotar de color a una entidad sin tener que dar color a los vértices de su malla. Este atributo se podrá consultar y modificar con sendos métodos `color` y `setColor`. El constructor recibirá también el color inicial como argumento, que tendrá como valor por defecto el blanco (`glm::vec4 color = 1`). Además, este constructor seleccionará el *shader* `simple`. Por último, sobrescribe el método `render()` con una definición similar a la de `EntityWithColors`, pero que que cargue el color en la GPU con `mShader->setUniform("color", mColor)`.

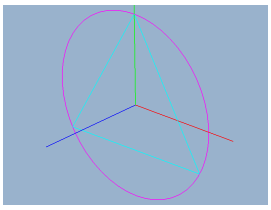
Apartado 4

Define la clase `RegularPolygon` que hereda de `SingleColorEntity` 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, a modo de ejemplo, un heptágono regular; pero debería ser válido para cualquier polígono.



Apartado 5

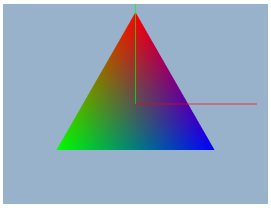
Añade a la escena un triángulo cian y una circunferencia magenta como objetos de la clase `RegularPolygon`, tal como se muestra en la figura.



Escena 2: Polígonos con relleno

Apartado 6

Define la clase `RGBTriangle` que hereda de `EntityWithColors` 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 `RegularPolygon`. Añade uno de estos triángulos a la escena.



Apartado 7

Core Profile no admite `glPolygonMode` diferenciado para la cara delantera y trasera porque es un caso particular del *cullling*.

Utilizando *culling*, redefine el método `render()` para 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:

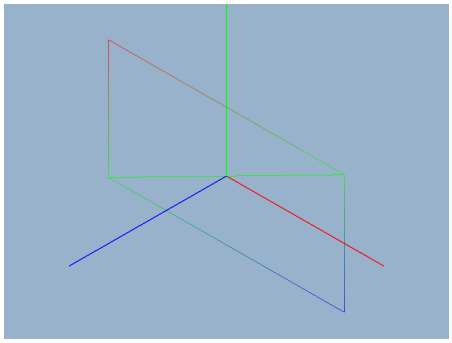
```
static Mesh* generateRectangle(GLdouble w, GLdouble h)
```

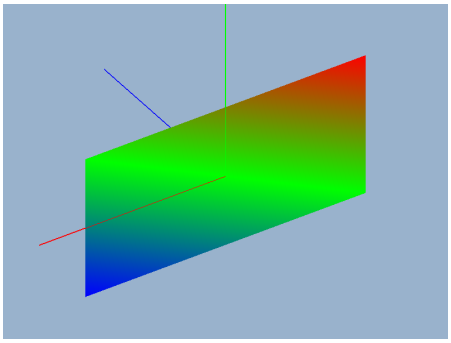
que genera los cuatro vértices del rectángulo centrado en el origen, sobre el plano $Z = 0$, de ancho w y alto h. Utiliza la primitiva `GL_TRIANGLE_STRIP`.

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 `EntityWithColors`, y añade una entidad de esta clase a la escena. Utilizando *culling*, redefine su método `render()` para establecer que los triángulos se rellenen por la cara **BACK** y se muestren con líneas, por la cara **FRONT**.



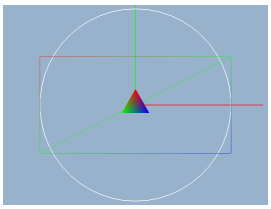


Apartado 9

Refactoriza el código para que cada escena sea una subclase de la clase genérica Scene. Para ello, convierte en virtual el método `init` de Scene y define una clase `Scene0` como subclase de Scene que defina su método `init` para la escena inicial. Ajusta la inicialización de `IG1App` para que construya una escena de tipo `Scene0`.

Apartado 10

Construye una escena 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 5.



Apartado 11

Coloca el triángulo RGB de la escena **1** en el punto **(R, 0)**, siendo R el radio de la circunferencia de esa escena.

Apartado 12

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 virtual `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 13

Sobrescribe el método `update()` en la clase `RGBTriangle` de forma que el triángulo de esta clase de la escena **1**, rote en horario sobre sí mismo a la par que lo hace en anti horario sobre la circunferencia.

Apartado 14

Implementa la actualización continua de la escena invocando periódicamente el método `update` de `IG1App`. Para ello, declara en dicha clase una constante `FRAME_DURATION`, una variable booleana `mUpdateEnabled` y una variable `mNextUpdate` de tipo `double`. En el bucle del método `run`, cuando `mUpdateEnabled` sea cierto, utiliza la función `double glfwGetTime()` para obtener el tiempo actual y llamar al método `update` cada `FRAME_DURATION` segundos. En ese caso, en lugar de `glfwWaitEvents` habrá que usar

```
void glfwWaitEventsTimeout(double timeout);
```

con el tiempo restante para llegar a `mNextUpdate`, que se irá actualizando oportunamente.

Haz que `mUpdateEnabled` se active y desactive con el evento de teclado **'U'**.

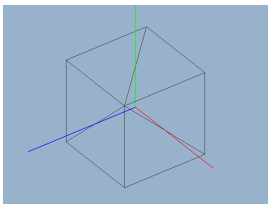
Escena 3: Cubo con color

Apartado 15

Define el método:

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

que construye la malla de un cubo (hexaedro) con arista de tamaño `length`, centrado en el origen. Define la clase `Cube` que hereda de `SingleColorEntity`, 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.

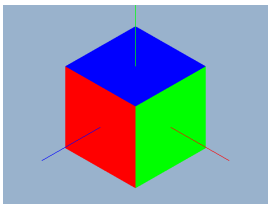


Apartado 16

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

```
static Mesh* generateRGBCubeTriangles(GLdouble length)
```

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



Apartado 17

(Opcional) Programa el método `update()` de la clase `RGBCube` tal como se muestra en la grabación "*demo de la escena 2*".

Entrega II Apartados del 18 al 37
Fecha de entrega: 13 de marzo de 2025

Instrucciones de la práctica

Para llevar a cabo este trabajo se tendrá en cuenta lo siguiente:

- Esta práctica se compone de una única escena con diversos elementos.
- Para realizar cada escena es necesario implementar una serie de pasos, en forma de apartados.
- El código que se desarrolle para cada escena deberá ser reutilizable por las escenas posteriores, incluidas en esta u otra práctica.
- Se tendrá especial atención a la calidad del código: inexistencia de fugas de memoria, código bien organizado, comentado y legible, se siguen los principios de la programación orientada a objetos, entre otros.

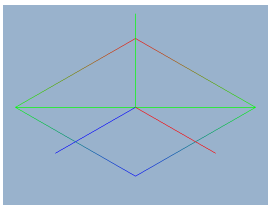
Rúbrica de evaluación

Criterio 1	**Pesos**
Suelo con textura	1.5
Caja con textura	1.5
Estrella doble con textura	1.5
Cristalera	1.5
Foto	1.5
Calidad del código	1.5
Ausencia de fugas	1
Total	**10.0**
Opcionales	+1
Defensa	-1.5

Escena 4: Texturas

Apartado 18

Define la entidad `Ground` como subclase de `EntityWithColors` cuyos objetos se renderizan como rectángulos centrados en el origen que descansan sobre el plano $Y = 0$. En la constructora, utiliza la malla `generateRectangle()` definida más arriba y establece la matriz de modelado para que el suelo se muestre siempre en posición horizontal.

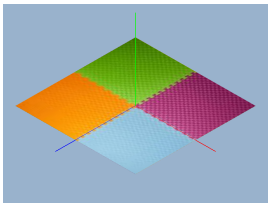


Apartado 19

Define la entidad `EntityWithTexture` para renderizar objetos con textura. Ha de guardar un atributo protegido `mTexture` de tipo `Texture*` con su textura y `mModulate` de tipo `bool` para indicar si se modulará la imagen con el color de los vértices (por defecto a `false`). El constructor seleccionará el shader *texture* y su método `render` tomará como referencia el de `EntityWithColors`, fijará el atributo uniforme `modulate` del shader y llamará a `bind` y `unbind` antes y después de renderizar la malla (solo si la textura es no nula).

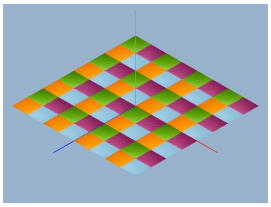
Apartado 20

Define en la clase `Mesh` el método `static Mesh* generateRectangleTexCor(GLdouble w, GLdouble h)` que añada coordenadas de textura al rectángulo para mostrar la imagen completa sobre él. Cambia la clase madre de `Ground` a `EntityWithTexture`, adapta su constructora y modifica su método `render` para que se muestre el suelo con la textura de la captura adjunta.



Apartado 21

Generaliza el método del apartado anterior a `static Mesh* generaRectangleTexCor(GLdouble w, GLdouble h, GLuint rw, GLuint rh)` para generar coordenadas de textura que embaldosen el suelo con la imagen, repitiéndola rw veces a lo ancho y rh veces a lo alto. En la captura, $rw = rh = 4$.

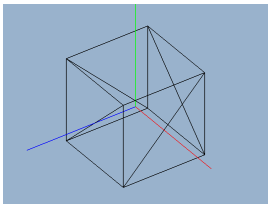


Apartado 22

Define el método `static Mesh* generateBoxOutline(GLdouble length)` que genera los vértices del contorno de un cubo, sin tapas, centrado en los tres ejes, con lado de tamaño longitud. Utiliza la primitiva `GL_TRIANGLE_STRIP`. Recuerda que el número de vértices, con esta primitiva, es 10: los 8 del cubo más 2 repetidos para cerrar el contorno.

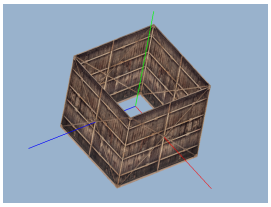
Apartado 23

Define la clase `BoxOutline` que hereda de `SingleColorEntity` y cuyos objetos se renderizan como cubos sin tapas, usando la malla del apartado anterior. En la captura se muestra el contorno de una caja, con las caras frontales y traseras en modo línea.



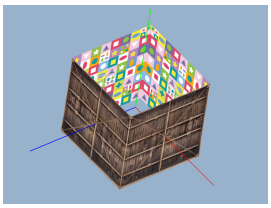
Apartado 24

Añade coordenadas de textura a la malla del contorno de una caja, definiendo el método `static Mesh* generateBoxOutlineTexCor(GLdouble length)`, y cambia la clase madre de `BoxOutline` a `EntityWithTexture`. Haz que la escena contenga el contorno de una caja con la textura que se muestra en la captura adjunta, repetida por las caras del contorno.



Apartado 25

Modifica el método `render()` de la clase `BoxOutline` de forma que se pueda renderizar la caja con dos texturas, una para el exterior y otra para el interior. Añade para ello un atributo de tipo `Texture*` (además del heredado) y utiliza apropiadamente el *culling*.

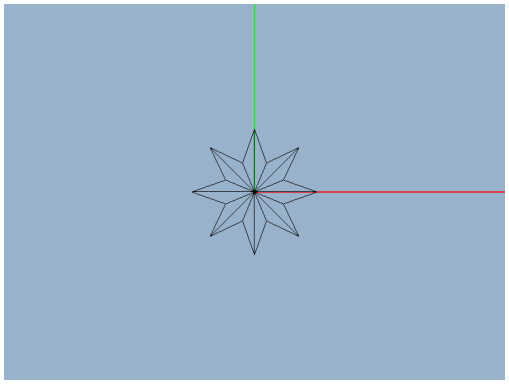


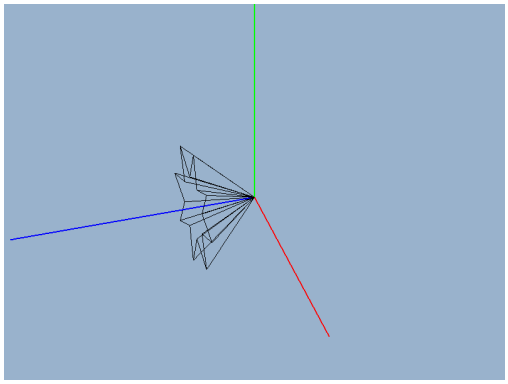
Apartado 26

Define el método:

```
static Mesh* generateStar3D(GLdouble re, GLuint np, GLdouble h)
```

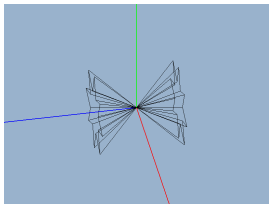
que genera los vértices de una estrella de `np` puntas situadas en los puntos de una circunferencia de radio exterior `re` centrada en el plano $Z = h$, como la que se muestra en las capturas. Utiliza la primitiva `GL_TRIANGLE_FAN` tomando como primer vértice el origen $(0, 0, 0)$. Los puntos internos se encuentran en una circunferencia de radio $ri = re/2$.





Apartado 27

Define la clase `Star3D` que hereda (provisionalmente) de `SingleColorEntity` y cuyos objetos se renderizan en estrellas como las mostradas. Sobrescribe el método `render()` en esta clase de forma que se muestren no una sino dos estrellas unidas por el origen, tal como se muestra en la captura.



Apartado 28

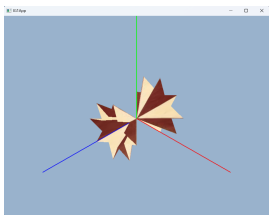
Define el método `update()` de la clase `Star3D` de forma que las dos estrellas enfrentadas roten coordinadamente sobre su eje Z a la vez que giran sobre su eje Y.

Apartado 29

Define el método:

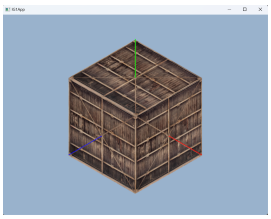
```
static Mesh* generateStar3DTexCor(GLdouble re, GLuint np, GLdouble h)
```

que genera coordenadas de textura para la malla de una estrella. Cambia la clase madre de `Star3D` a `EntityWithTexture` y adapta el método `render()` para renderizar la estrella con textura tal como se muestra en la captura, con una estrella de 8 puntas.



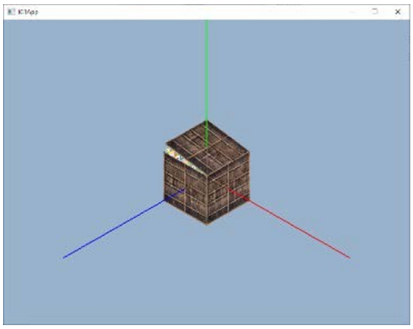
Apartado 30

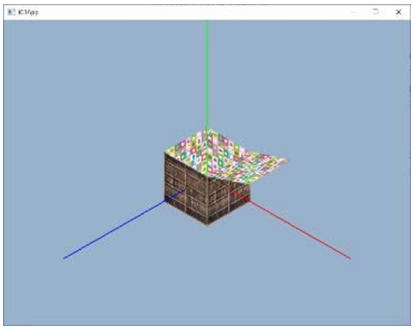
(Opcional) Define la clase `Box` mediante la malla de un contorno de caja junto con dos mallas de rectángulo más, una para la tapa y otra para el fondo. La renderización de una caja se muestra en la captura. Aunque no se ven, las caras interiores de la caja tienen todas la textura interior del contorno de una caja.



Apartado 31

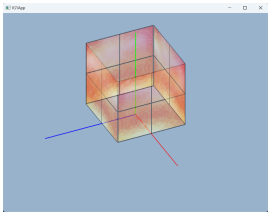
(Opcional) Define el método `update()` de la clase `Box` de forma que la tapa se abra 180º para después volver a cerrase y así sucesivamente.





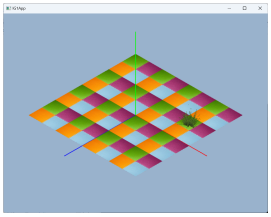
Apartado 32

Define la clase `GlassParapet` cuyos objetos se renderizan en contornos de caja con una textura traslúcida en todas sus caras, tal como se muestra en la captura adjunta.



Apartado 33

(Opcional) Define la clase `Grass` cuyos objetos se renderizan encima del suelo, en una esquina, mostrando la textura anterior (con fondo transparente), rotada y renderizada tres veces. Para tener en cuenta la transparencia, utiliza el shader de fragmentos `texture_alpha`, que puedes cargar con `Shader::get("texture:texture_alpha")`.



Apartado 34

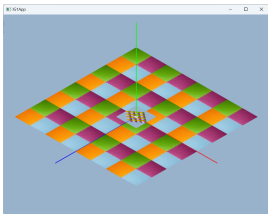
En la clase `Texture`, define un método

```
void loadColorBuffer(GLsizei width, GLsizei height, GLuint buffer=GL_FRONT)
```

que cargue el buffer de color (frontal o trasero) dado por el tercer argumento, como una textura de dimensiones dadas por los parámetros primero y segundo.

Apartado 35

Define la clase `Photo` que hereda de `EntityWithTexture` y cuyos objetos se renderizan en un rectángulo centrado sobre el suelo, tal como se muestra en la captura adjunta. El rectángulo tiene adosada una textura obtenida de la imagen que carga el método del apartado anterior. Define el método `update()` de esta clase de forma que su atributo `mTexture` se actualice a la textura de este rectángulo.



Apartado 36

(Opcional) Implementa el evento de teclado **F** que guarda la imagen de la foto hecha como en el apartado anterior, como fichero `.bmp`.

Apartado 37

Define una escena que contenga un suelo, una caja con su tapa que se abre y se cierra, situada en una esquina del suelo, una estrella encima de la caja, una cristalera que rodea el suelo, unas hierbas y una foto. Evidentemente, no es obligatorio que aparezcan los apartados opcionales.