Práctica 2: Modelos poligonales

Objetivos

- Entender la representación de mallas de triángulos
- Saber calcular las normales de los triángulos y los vértices
- Saber leer, representar y visualizar una malla de triángulos representada en un archivo PLY
- Saber crear mallas de objetos de revolución a partir de la poligonal del perfil

Código inicial

Partiremos del código creado en la práctica 1.

Funcionalidad a desarrollar

- **Representación de mallas de triángulos**. Se creará una clase para representar mallas de triángulos, con posibilidad de almacenar normales por vértice y por triángulo.
- **Objetos ply**. Se creará un constructor de mallas de triángulos que lea la información de la malla de un archivo ply.
- **Superficies de revolución**. Se creará una subclase de malla de triángulo para representar superficies de revolución.
- **Cálculo de normales**. Para ambos tipos de objetos se deberán calcular las normales de cara y de vértice.
- **Dibujo en modos FLAT y SMOOTH** El método de dibujo debe permitir visualizar las mallas tanto en modo FLAT como en modo SMOOTH.
- **Creación de escena**. Se incluirá en la escena al menos una superficie de revolución y dos ply uno dibujado en modo FLAT y el otro en modo SMOOTH.

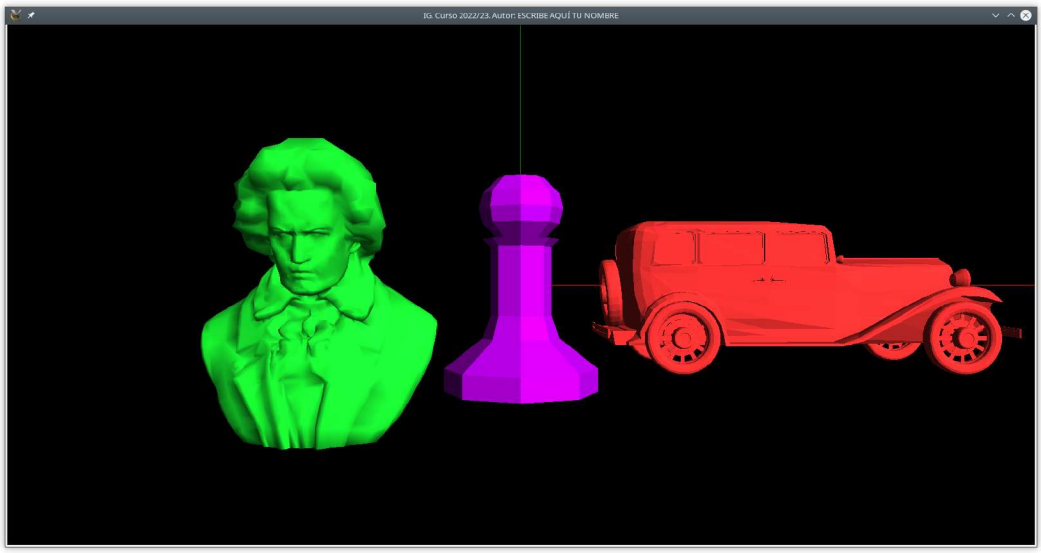

Figura 15: Escena creada en la práctica 2

La Figura 15 muestra un posible resultado de la práctica. La escena creada debe contener dos modelos PLY (cada uno dibujado con un modo de sombreado diferente) y una superficie de revolución.

Desarrollo

Representación de la malla de triángulos

Crea estructuras de datos para representar mallas de triángulos que te permitan identificar la malla con una variable (para facilitar el pasarla como argumento a la función de dibujo de mallas que crearás mas adelante en esta práctica). La malla debe contener al menos los vértices, los triángulos y las normales de vértice y de triángulo.

Si quieres hacer el código orientado a objetos puedes crear una clase malla virtual heredando de Objeto3D.

Lectura de ply

Vamos a incluir funcionalidad para que se puedan leer modelos de un archivo en nuestras mallas. Si has creado una clase para representar mallas puedes crear una subclase y programar un constructor que cree la malla a partir del contenido de un archivo que le pasas como argumento.

Las mallas las leeremos de archivos PLY. PLY es un formato para el almacenamiento de modelos poligonales en fichero. Las siglas proceden de "Polygon File Format", ha sido diseñado por la universidad de Stanford.

Un fichero ply puede contener información en formato ascci o binario. En cualquier

caso tendrá una cabecera en ascci que indica el tipo de datos que contiene. La cabecera determina además como se estructura la información, la geometría que contiene, tipos de datos etc.

El formato permite guardar vértices, polígonos y atributos de vértices (normales, coordenadas de textura,..).

Es posible descargar modelos 3D en formato PLY de diferentes web (p.e. en 3dvia o en Robin.

Hay muchas funciones publicadas para leer archivos PLY. En el código de prácticas tienes incluido el lector de PLYs de Carlos Ureña. Consta de dos archivos:

- file_ply_stl.h : declaración de las funciones ply::read y ply::read_vertices.
- file_ply_stl.cc : implementación.

Para leer un archivo PLY basta con llamar a la función *ply* :: *read*, tal como se muestra en el siguiente ejemplo:

```
#include <vector>
#include "file_ply_stl.h"

...
std::vector<float> vertices_ply ; // coordenadas de vertices
std::vector<int>   caras_ply ;    // indices de vertices de triangulos

...
ply::read( "nombre-archivo", vertices_ply, caras_ply );
...
```

Tras la llamada se obtienen en *vertices_ply* las coordenadas de los vértices ($3n$ flotantes en total, si hay n vértices en el archivo) y *encaras_ply* los índices de vértices de los triángulos ($3m$ enteros, si hay m triángulos en el archivo).

Cálculo de normales

Para visualizar las mallas con iluminación necesitamos calcularle las normales. Añadiremos una función para calcular las normales de cara y de vértice.

Esta funcionalidad puede ser un método que puedes llamar después desde el constructor después de cargar el modelo PLY.

Para calcular la normal de una cara triangular constituida por vértices P_0, P_1 y P_2 ordenados en sentido antihorario, calculamos el producto vectorial de los vectores $\overrightarrow{(P_0,P_1)}$ y $\overrightarrow{(P_0,P_2)}$ (ecuación 1) y dividimos el resultado por su módulo para obtener un vector perpendicular y de módulo unidad (ecuación 2) .

$$\vec{N_0} = (\overrightarrow{(P_0,P_1)} x \overrightarrow{(P_0,P_2)}) \tag{1}$$

$$\vec{N} = \frac{\vec{N_0}}{modulo(\vec{N_0})} \tag{2}$$

Las normales de los vértices se pueden calcular sumando las normales de las caras que comparten el vértice, y normalizando el vector resultante (observa que el resultado no es la media de los vectores normales, ya que la media no será un vector normalizado).

Dado que en nuestra estructura de datos tenemos enlaces Cara-Vértice, debemos hacer la suma de las normales de los vértices iterando en la lista de caras:

```
Inicializar normales de todos los vertice a (0,0,0)
Para cada cara
  sumar su normal a sus tres vertices
Para cada vertice
  Normalizar su normal
```

Tanto al calcular las normales de la cara como las de los vértices se debe comprobar que el módulo del vector es mayor que cero antes de hacer la división por el módulo.

Dibujo con sombreado plano y suave

Para dibujar el modelo con sombreado plano (cálculo de iluminación por caras) usamos:

glShadeModel(*GL_FLAT*);

antes de comenzar a dibujar el modelo. La iluminación de la cara se calcula con la normal que se haya pasado a OpenGL antes del último vértice de la cara. Por tanto, tenemos que dar la normal de cada cara (usando $glNormal3f(nv,ny,nz)$) antes de que se haya enviado el último vértice de la cara.

Para dibujar con sombreado suave (cálculo de iluminación por vértice) usamos:

glShadeModel(*GL_SMOOTH*);

y damos la normal de cada vértice justo antes de enviar el vértice.

Para simplificar el código puedes hacer dos métodos de dibujo de las mallas, una para cada modo de sombreado.

Superficies de revolución

Se desarrollará un método para la generación procedural de una malla obtenida por revolución de un perfil poligonal alrededor del eje Y. Dicho algoritmo tiene como parámetros de entrada el perfil, y el número de copias del mismo que servirán para crear el objeto.

Puedes crear una subclase de Malla para la que este método sea el constructor.

El perfil de entrada es una secuencia de *m* vértices en 3D en el plano XY (su coordenada z es cero). El perfil inicial se puede leer de un fichero PLY cuyo contenido sólo ha de tener las coordenadas de los vértices. Este fichero PLY puede escribirse manualmente, o bien se puede usar el que se proporciona en el material de la práctica. Para leer el perfil puedes usar la función *ply* :: *read_vertices*

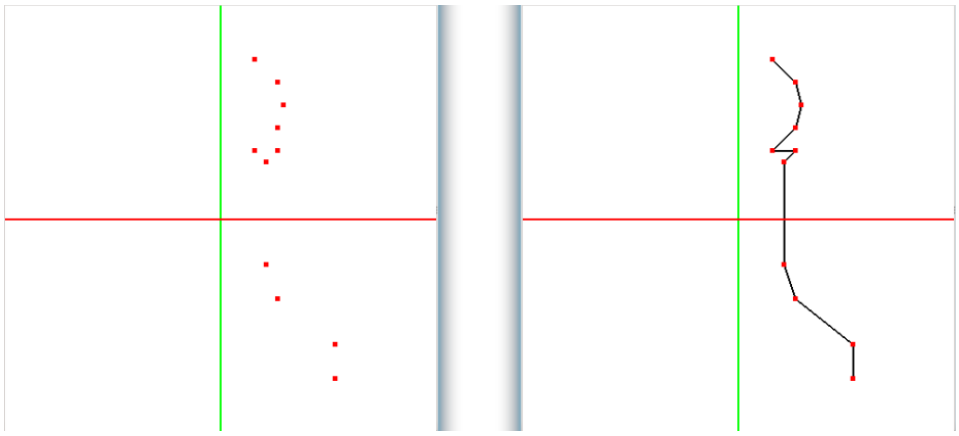

Figura 16: Ejemplo de perfil (puntos a la izquierda, poligonal a la derecha).

```
#include <vector>
#include "file_ply_stl.h"

...
std::vector<float> vertices_ply ; // Coordenadas de vertices
...
ply::read_vertices( "nombre-archivo", vertices_ply );

...
```

Algoritmo de creación

Para la creación de un objeto de revolución, lo más fácil es crear en primer lugar la lista de vértices (partiendo del perfil original) y en segundo lugar la lista de triángulos. Para simplificar la creación de la lista de triángulos se pueden duplicar los vértices del perfil original (de hecho hacerlo simplificará la aplicación de una textura en la práctica 4).

Supongamos que el perfil tiene m vértices, que nombramos como $(p_0,..,P_{m-1})$ Asumimos que los vértices se dan de abajo hacia arriba (si cambiase el sentido cambiaría la orientación de las caras). Normalmente los vértices tienen coordenadas X estrictamente mayor que cero, pero el algoritmo funciona bien si algún vértice tiene coordenada X igual a cero, en este caso se crearían triángulos degenerados (sin área), que OpenGL ignora durante la rasterización. La figura 16 muestra un ejemplo de perfil.

De ese perfil original haremos n replicas rotadas (a cada una de ellas la llamamos una instancia del perfil). El valor de n debe ser mayor que tres. Cada uno de estas instancias del perfil forma un ángulo de $2\pi/(n-1)$ radianes con la siguiente o anterior.

Las instancias del perfil se numeran desde 0 hasta $n-1$, por tanto, la i-ésima instancia forma un ángulo de $2\pi i/(n-1)$ radianes con la instancia número 0. Las instancias0 y $n-1$ tienen sus vértices en la mismas posiciones que el perfil original. Se van a crear nm vértices en total. Dichos vértices se insertarán en la lista de vértices por instancias del perfil (es decir,

todos los vértices de una misma instancia aparecen consecutivos), además las instancias se almacenan en orden (empezando en la instancia 0 hasta la $n-1$).

Por tanto,en la lista final de vértices, el j-ésimo vértice de la i-ésima instancia tendrá un índice en la lista igual a $im+j$ (donde i va desde 0 hasta $n-1$ y j va desde 0 hasta $m-1$).

Para crear los vértices, por tanto, bastará con hacer un bucle doble que recorre todos los pares (i, j) y en cada uno de ellos crea el vértice correspondiente y lo inserta al final de la lista de vértices.

El pseudo-código, por tanto, para la creación de la lista de vértices será como sigue: Partimos de la lista de vértices vacía.

```
Para cada i desde 0 hasta n-1 (ambos incluidos)
  Para cada j desde 0 hasta m-1 (ambos incluidos)
   q = Rotacion de Pj  2iPI/(n-1) radianes respecto al eje Y
   Agregar q al final de la lista de vertices.
```

Para crear la lista de caras basta hacer un bucle con un índice i que recorre las instancias. Para cada instancia recorremos sus vértices con otro bucle excepto el último de ellos. Por cada vértice visitado se insertan en la lista de caras dos triángulos adyacentes (que comparten la arista diagonal).

Es decir, pseudo-código para la lista de triángulos visita todos los vértices (excepto el último de cada instancia y los de la última instancia), y crea dos triángulos nuevos que son adyacentes a ese vértice:

```
Partimos de la lista de triangulos vacia
Para cada i desde 0 hasta n-2 (ambos incluidos)
  Para cada j desde 0 hasta m-2 (ambos incluidos)
    Sea k = i m + j
      Agregar triangulo formado por los indices k, k+m y k+m+1
      Agregar triangulo formado por los indices k, k+m+1 y k+1
```

Una vez creada la malla debemos calcular las normales. Podemos usar la misma función usada para las mallas creadas a partir de archivos PLY.

Creación de la escena

El programa final debe crear una escena como la de la figura 15.

Evaluación

La evaluación de la práctica, sobre 10 puntos, se hará del modo siguiente:

- Dos puntos por la estructura de datos.
- Dos puntos por la creación de mallas a partir de archivos PLY.
- Dos puntos por la creación de mallas de superficies de revolución.

- Dos puntos por el cálculo de normales.
- Dos puntos por la visualización en modos FLAT y SMOOTH.
- Hasta dos puntos por crear figuras adicionales (p.e. barrido lineal).

Temporización

Esta práctica se debe realizará en tres sesiones de prácticas:

Grupo C1 Jueves 29/09/22, 06/10/22 y 13/10/22

Grupo C2 Miércoles 28/09/22, 05/10/22 y 19/10/22