Prácticas de Informática Gráfica

Grupos C y D (Curso 2024/2025)

Germán Arroyo y Juan Carlos Torres

2024-06-06

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.
- **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.

La Fig. 1.1 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.

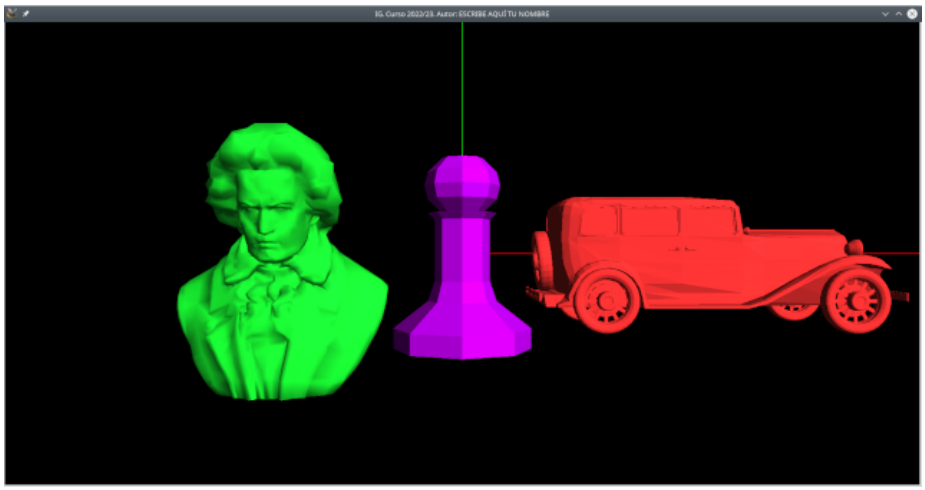

Figura 1: Escena creada en la práctica 2

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 ASCII o binario. En cualquier caso tendrá una cabecera en ASCII 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"

...

// coordenadas de vértices
std::vector<float> vertices_ply ;

// índices de vertices de triángulos
std::vector<int> caras_ply ;

...

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 `en caras_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 [eqN0]) y dividimos el resultado por su

módulo para obtener un vector perpendicular y de módulo unidad (ecuación [eqN]) .

$$\vec{N_0} = (\overrightarrow{(P_0, P_1)} \times \overrightarrow{(P_0, P_2)})$$

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

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:

1. Inicializar normales de todos los vértices a $(0, 0, 0)$
2. Para cada cara:
   - Sumar su normal a sus tres vértices
3. Para cada vértice:
   - 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.

Creación de la escena

El programa final debe crear una escena mostrando los elementos creados(ver Fig. 1.1).

Temporización

Se recomienda realizar esta práctica en dos sesiones.