OpenGL lab

Collada Object Viewer

Formato de archivo Collada (.dae)

Collada es un formato de archivo abierto que usa XML para guardar modelos 3D: primitivas, vértices, normales, coordenadas de textura, animaciones, etc... En este laboratorio vamos a hacer un parser de la información básica que viene en un archivo .dae tomando algunos atajos para evitar complicarlo demasiado. Probaremos a cargar los ficheros:

- EM208_heavy.dae
- Venom.dae

Usando el segundo modelo como ejemplo, la parte del árbol XML que nos interesa (en negrita) es:

```
...
<COLLADA xmlns="http://www.collada.org/2005/11/COLLADASchema"
version="1.4.1">
...
  <library_images>
    <image id="Image">
      <init_from>file:///Venom_D.tga</init_from>
    </image>
...
  <library_geometries>
    <geometry id="Geometry" name="venom">
      <mesh>
        <source id="GeometrySource" name="venomPos">
          <float_array id="GeometrySource-array" count="13626">-
0.084239 -0.334797 3.6898 -0.053587...</float_array>
          ....
        </source>
        <source id="GeometrySource" name="venomNrm">
        <float_array id="GeometrySource1-array" count="13626">-
0.079075 -0.742499 -0.665163...</float_array>
          ....
        </source>
        <source id="GeometrySource" name="venomUV">
```

```
            <float_array id="GeometrySource2-array" count="9084">0.19873
0.383789 0.168091 0.353027...</float_array>
              ....
            </source>
            ...
            <triangles material="VisualMaterial0" count="7292">
              ...
              <p>0 1 2 3 4 5 6 7 8 6 8 9 9 8 10 9 10 11 12 13 ...</p>
            </triangles>
    ...
```

Lo que supondremos para simplificar el parsing:

- cada modelo tiene una única textura (*"COLLADA/library_images/image/init_from"*)
- tendrá 3 listas de números reales ("*COLLADA/library_geometries/geometry/mesh/source*")
  - una llamada "*Pos" con las coordenadas <x,y,z> (3 por vértice)
  - una llamada "*Nrm" con las normales <nx,ny,nz> (3 por vértice)
  - una llamada "*UV" con las coordenadas de textura <u,v> (2 por vértice)
- tendrá una lista de triángulos, cada uno con 3 índices en: "*COLLADA/library_geometries/geometry/mesh/triangles/p*".

1 Crear una nueva clase ColladaModel

Añadir al proyecto una nueva clase llamada *ColladaModel* que hereda de *GraphicObject3D*. El constructor de esta clase tomará el nombre del modelo a cargar como parámetro. Además, implementará el método abstracto *draw()* heredado de *GraphicObject3D*.

1.1 Cargar el fichero XML

El primer paso será crear el constructor de la clase. En este constructor, tenemos que leer el archivo XML y guardar la información que necesitamos para dibujar el modelo como atributos de la clase:

- std::vector<double> m_positions;
- std::vector<double> m_normals;
- std::vector<double> m_texCoords;
- int textureId; //usaremos SOIL para cargar la textura y guardaremos el identificador que nos devuelva

Para cargar el archivo XML usaremos la librería tinyxml2 que ya está incluída en la solución (incluir *"tinyxml2/tinyxml2.h"* y *"Debug/tinyxml2.lib"*). Usar la librería para cargar el fichero y recorrer el árbol es relativamente sencillo:

```
tinyxml2::XMLDocument doc;
doc.LoadFile("modelo.dae");
tinyxml2::XMLElement* pRoot= doc.FirstChildElement("COLLADA");
tinyxml2::XMLElement* pLibraryImages= pRoot->FirstChildElement("library_images");
```

Para coger el contenido de un XMLElement usaremos el método *const char* XMLElement::GetText()*. Para hacer el parsing de una lista de índices y vértices añadiremos estos dos métodos a la clase ColladaModel:

```
void ColladaModel::parseXMLFloatArray(tinyxml2::XMLElement *pFloatAr-
ray, std::vector<double> &vector)
{
  char* pCharArray = (char*) pFloatArray->GetText();
  char* nextToken;
  char* pt = strtok_s(pCharArray, " ", &nextToken);
  while (pt != 0)
  {
    vector.push_back( atof(pt));
    pt = strtok_s(0, " ", &nextToken);
  }
}
void ColladaModel::parseXMLIntArray(tinyxml2::XMLElement *pFloatAr-
ray, std::vector<int> &vector)
{
  char* pCharArray = (char*)pFloatArray->GetText();
  char* nextToken;
  char* pt = strtok_s(pCharArray, " ", &nextToken);
  while (pt != 0)
  {
    vector.push_back( atoi(pt));
    pt = strtok_s(0, " ", &nextToken);
  }
}
```

Llamaremos a estos dos métodos pasándoles el nodo que contiene la lista de números reales/enteros y el vector al que hay que pasárselo. Por ejemplo:

parseXMLFloatArray(pFloatArray,m_positions);

1.2 Dibujo del modelo

Al dibujar tenemos que pasarle a OpenGL triángulos usando el array de índices: 3 índices por triángulo. Con esos índices accederemos a las posiciones, normales y coordenadas de textura, usando las mismas funciones que para dibujar los cubos.

Cosas que hay que tener en cuenta a la hora de dibujar:

- En estos dos modelos, la segunda coordenada de textura hay que invertirla (0.3 -> 1-0.3= 0.7) porque el origen de coordenadas no se corresponde con el de OpenGL

- En estos dos modelos, el eje que apunta hacia "arriba" es el z, en vez del y (OpenGL), así que para ver un modelo correctamente, hará falta girarlo 270 grados.