COSC422 Advanced Computer Graphics
Programming Exercise 14
Assimp 3D Model Loader

This programming exercise aims to familiarise you with the scene graph structure consisting of a hierarchy of transformations used by model loading libraries such as Assimp.

Assimp:

Please download and install Assimp (version 5) from

https://www.assimp.org/

Information on data structures and classes used by Assimp, as well as the API documentation can be found at

https://assimp-docs.readthedocs.io/en/master/about/introduction.html

Assimp_ModelLoader.cpp:

The program `Assimp_ModelLoader.cpp` provides a simple implementation of a 3D model loader using Assimp. The file `assimp_extras.h`. contains a set of helper functions useful in such implementations.

The function `loadModel()` creates the scene object for the input model. The name of the input model file is specified in the `initialize()` function. A scene object is generated when a model is loaded.

The function **`render()`** is the main part of the model loader. This recursive function is used to traverse the scene graph from its root node. The variable `node` (the second parameter of the function) represents the current node in the node hierarchy. Each node stores an array of mesh indices given by (Slide [9]-11)

`meshIndex = node->mMeshes[i],`
`i` = 0 … (`node->mNumMeshes`)−1.

Using these indices, the mesh objects are retrieved from the scene object:

`mesh = scene->mMeshes[meshIndex]` (see Slide [9]-10).

Each mesh contains a single material index given by `mesh->mMaterialIndex`.

The `render()` function iterates over all meshes of the current node (`for (int n=0;…)`) and draws all polygons of each mesh (`for (int k=0;…)`). After processing the current node, the `render()` function is recursively called to descend to the child nodes. The transformation hierarchy in the scene graph is directly translated into a nested structure of glPushMatrix()-glPopMatrix() blocks.

The `display()` function renders the model as generated by the `render()` function. The bounding box of the model is used to scale the model to fit within the display window.

Most mesh model definitions use the *z*-axis as the primary axis for modelling, and require a –90 degs rotation about the *x*-axis. By default, the program displays models after this transformation. Use the keyboard input '1' to turn this rotation on or off. The print functions in `loadModel()` may be uncommented to get the output of scene and node parameters.

The program includes a function `loadGLTextures()` to load images stored in various formats as textures. The Developer's Imaging Library (DevIL/OpenIL) is used for this purpose. The function `loadGLTextures()` visits all material objects of the scene and checks if any of them has any texture file information stored in them. If a material object contains a texture file name, the function loads that texture and associates the texture id with the material id using a hash map (`texIdMap`). The `render()` function checks if the current mesh has texture coordinates and enables texturing as needed (`if (mesh->HasTextureCoords... )`

If a model file uses textures, it will have the relative path and file names of textures stored in material definitions. All four models provided with this exercise use textures (Fig. 1).

			
dwarf.x	Truck.3ds	Ana.x	knight.blend

Fig. 1.

Model files can contain incorrect texture names or invalid file paths. The "knight.blend" model specifies the filename for the texture as `//Knight.png`. In such cases where a texture could not be loaded using the specified file name or path, the file name may be manually set in the `loadGLTextures()` function as shown below:

```
if (ilLoadImage((ILstring)"Knight.png"))
{
    ...
```

[9]: COSC422 Lecture Slides: Lec09_SceneGraphs.pdf