ASSIMP

With complicated shapes (e.g. houses, vehicles, human-like characters), it would be incredibly tedious to try and manually define all the vertices, normals, and texture coordinates. Instead, we want to **import** these models that were created in some 3D modeling tool into our application.

3D modeling tools allow artists to create complicated shapes and apply textures to them via a process called **uv-mapping**. The tools then automatically generate all the vertex coordinates, vertex normals, and texture coordinates while exporting them to a model file format we, as developers, can use.

We need to parse these exported model files and extract all the relevant information so we can store them in a format that OpenGL understands.
A common issue is that there are many different file formats where each exports the model data in its own unique way.

- Example: **Wavefront .obj** only contains model data with minor material information like model colors and diffuse/specular maps.
  - Easy to parse
- Example: The XML-based **Collada file format** is extremely extensive and contains models, lights, many types of materials, animation data, cameras, complete scene information, and more.

Take a look at the wavefront .obj wiki to see how the data is formatted. This should give you a basic perception of how model file formats are generally structured.

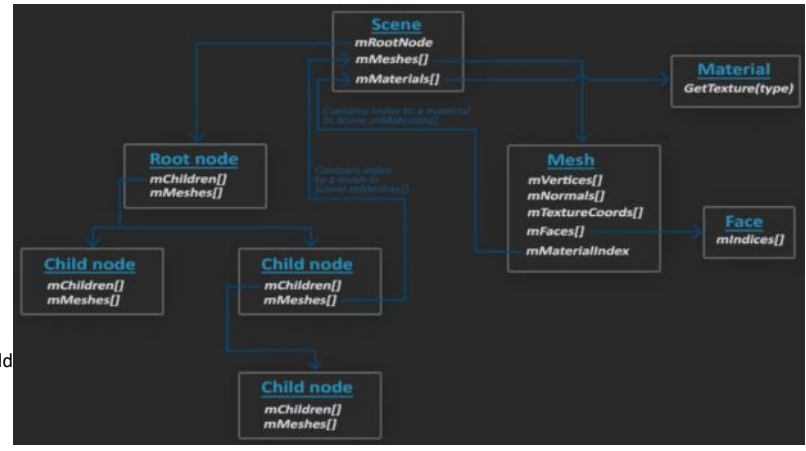
If we wanted to import a model from multiple different file formats, we'd have to write an importer for each different file format. Instead, we'll use the Assimp library.

A Model Loading Library

Assimp, which stands for Open Asset Import Library, is a popular model importing library that is able to import many different model file formats (and export to some as well) by loading all the model's data into Assimp's generalized data structures.

- As soon as Assimp has loaded the model, we can retrieve all the data we need from Assimp's data structures, abstracting us from all the different file formats out there.

To the right is a simplistic model of Assimp's structure.



Assimp loads imported models into a scene object that contains all the data of the imported model/scene. Each node contains indices to data stored in the scene object where each node can have any number of children.

- The Scene object contains all the data of the model/scene, like materials and meshes. It also contains a reference to the root node of the scene.
- The Root node of the scene may contain children nodes (like all other nodes) and could have a set of indices that point to mesh data in the scene object's mMeshes array.
  - The scene's mMeshes array contains the actual Mesh objects, the values in the mMeshes array of a node are only indices for the scene's meshes array.
- A Mesh object itself contains all the relevant data required for rendering.
  - e.g. Vertex positions, normal vectors, texture coordinates, faces, and the material of the object.
- A mesh contains several faces. A Face represents a render primitive of the object (triangles, squares, points). A face contains the indices of the vertices that form a primitive.
  - Because the vertices and the indices are separated, this makes it easy for us to render via an index buffer.
- A mesh also links to a Material object that hosts several functions to retrieve the material properties of an object.
  - e.g. Colors and/or texture maps (like diffuse and specular maps).

What we want to do:

1. Load an object into a Scene object.
2. Recursively retrieve the corresponding Mesh objects from each of the nodes (we recursively search each node's children).
3. Process each Mesh object to retrieve the vertex data, indices, and its material properties.

The result is a collection of mesh data that we want to contain in a single model object.

NOTE: A single mesh is the minimal representation of what we need to draw an object in OpenGL (vertex data, indices, and material properties). A model (usually) consists of several meshes.

Building Assimp

Compile the Assimp libraries into a .lib file using CMake (like you did with GLFW in the Creating a Window chapter).

- Link to Assimp GitHub: Assimp v3.1.1 GitHub
- If CMake gives an error regarding a missing DirectX library, install the DirectX SDK here.
- If, while installing the DirectX SDK, you get an error code of s1023, uninstall the C++ Redistributable package(s) from your system and try installing the DirectX SDK again.
- Steps for Building Assimp into Project

Steps for Building Assimp into Project

Wednesday, April 13, 2022 8:21 PM

1. Download Assimp from the GitHub repository.
   a. There are multiple ways you can do this, but the simplest is to click the green "Code" button and select "Download Zip".
      i. Extract all the files.
2. Compile the library file using CMake.
   a. Go to the install directory for assimp, go into the ..\assimp-3.1.1\assimp-3.1.1\ directory (probably something like C:\Users\YOURUSER\Downloads\assimp-3.1.1\assimp-3.1.1), and create a new folder named "build".
   b. Open CMake and select ..\assimp-3.1.1\assimp-3.1.1\ as the location of the source code; ../assimp-3.1.1/assimp-3.1.1/build/ as the directory to build the binaries.
   c. Click Configure.
      i. If CMake gives an error regarding a missing DirectX library, install the DirectX SDK here.
      ii. If, while installing the DirectX SDK, you get an error code of s1023, uninstall the C++ Redistributable package(s) from your system and try installing the DirectX SDK again.
   d. Click Generate.
   e. Go to the ..\assimp-3.1.1\assimp-3.1.1\build\ directory and open the Assimp.sln solution file.
   f. Build the solution (ctrl+b). This generates the library file.
3. Move the necessary files over to your project.
   a. The assimpd.lib and assimpd.dll files are located at ..\assimp-3.1.1\assimp-3.1.1\build\code\Debug\.
      i. Copy the .lib file into your libraries folder for your project.
      ii. Copy the .dll file into the same directory as your application's executable (something like .. \PROJECT\x64\Debug\).
         □ The default configuration builds Assimp as a dynamic library, so we need to include the resulting DLL, as well as the application's binaries.
   b. Copy the assimp folder in the include directory, located at ..\assimp-3.1.1\assimp-3.1.1\include\, into your include directory for your project.
4. Link these files from your project.
   a. Open your solution file.
   b. Right-click the project in the Solution Explorer and select Properties.
   c. Go to Configuration Properties->Linker->Input and add assimpd.lib to the Additional Dependencies.
5. Try to include a header file from the assimp library in your project to make sure it works.