Assignment II, Deadline 29/11/2012

This assignment is part 1 of a multi-part assignment; the other parts will be built on top of this.

Your task: Import 3d models with Assimp & render them using OpenGL 3.3, with the glsl program wrapper class you wrote & vertex buffer objects. Your app should be capable of rendering the models in this rar file correctly.

Libraries: glew, sfml, assimp, glm

- Your application will be started via command line. It will receive the filepath of a 3d model, preceded by –m. A call to your application (myApp.exe)will look like this:

  myApp –m filepath

- parse the commandline args, and use Assimp to load the model in question (if it's not found, you can terminate immediately)

  loading a model in assimp can be done like this:

```
std::string filepath;

Assimp::Importer imp;

//this tells assimp to remove point & line primitives, so that
//triangles remain as the only primitive type

imp.SetPropertyInteger( AI_CONFIG_PP_SBP_REMOVE,
aiPrimitiveType_POINT | aiPrimitiveType_LINE );

//this tells assimp to scale the model so that all vertices lie in
//the [-1, 1] range

imp.SetPropertyInteger( AI_CONFIG_PP_PTV_NORMALIZE , 1 );

//for an overview of all postprocessing flags, see
//http://assimp.sourceforge.net/lib_html/postprocess_8h.html

const aiScene* scene = imp.ReadFile( filepath,
aiProcess_PreTransformVertices | aiProcess_JoinIdenticalVertices |
aiProcess_FindDegenerates | aiProcess_SortByPType |
aiProcess_Triangulate | aiProcess_RemoveComponent |
aiProcess_ValidateDataStructure );
```

  if you use the above flags, you will receive a pointer to an aiScene with at least one (but possibly more) aiMesh (es) . (If the import fails, this pointer will be null and you can terminate your application.) Each mesh is guaranteed to have a list of vertex positions (aiVector3D), and a list of aiFaces which are all triangles, since all other primitive types were explicitly removed with the help of the postprocessing flags. Create a vertex buffer for the vertex positions & build an index buffer from the the face list.

- Create a very basic pair of fragment shaders:

  - The vertex shader needs only 1 input attribute: the vertex position
  - Additionally, it should have a model-, view-, and projection matrix as uniform
  - Transform the vertex position with the help of your matrices
  - The fragment shader can just output each fragment in a hardcoded color for now (materials & shading are part of the next assignment)
- Use the wrapper class you created to load the shaders & create a program object.
- Draw the model using you shaders. Set up a a camera with the help of the glm::lookAt and glm::perspective functions. (look into the code snippets on the wiki page for a sample on how to do this).
- Rotate the model around the y axis based on the elapsed time (use sf::Clock)
- Allow the user to toggle on and off wireframe mode with the help of the ‘W’ key.