OpenGL Model Loader

By Kacper Mazur

# Which version Visual Studio, OpenGL/Vulkan:

* Visual studio 2019
* OpenGL

# How do I work it?

* Parses whole obj file
* Uses string streams to separate data out
* Everything gets stored in temp vectors of glm vecs 2 and 3;
* It keeps track of the file path and tries to find mtl file automatically
* During a face parsing, the algorithm will check if the faces are triangulated and if not, it will proceed to triangulate and output the new indices to temp vectors
* Vertices(positions) & normal get uncompressed with the newly generated indices;
* Then the vertices are inputted into a Vertex and then into a Vertex array inside the mesh (more info below)
* Now these Vertex structs will be put through a unordered map (hash table)
  + Hash table will store any unique vertex objects and with a value of vertices vector (starts of 0 and increments).
  + The unique index gets pushed to temp vector
  + Also the unique vertex get stored into another temp vector
  + If a Vertex is not unique than it will get pushed into index vector with a retrieved int value from the hash table

# How does your code fit together and how should a programmer navigate it:

* The Loader Out puts 3 structs and one contains all of them;
  + Material struct holds all the values from the mtl file
  + Vertex struct holds a glm::vecs of positions, texcoord and normals.
  + Mesh Struct holds a vector of Verex structs and Materials with the unique indices that got out putted form the hash table.
* Two functions that User is able to call Loader::LoadMesh(str) which will return a populated mesh struct (with the mtl files)
  + Loader::LoadMaterial(str) will out put a vector of materials which can easily get placed inside Mesh.Materials

# Extra Information:

* Loader is able to load multiple materials from a single mtl file
* Loader is unable to separate each internal mesh within a obj file (unable to batch render)
* Look at the ParseOBJ functions, it goes sequential through the whole parsers.
* Able to compress Verices upto 82% from 120k vertices to 21k