**STRUCTURE Prefer simple structure over complexity at this point**

Node:

* Transform matrix (array of 16 floats, 4x4 matrix)
* Vector<Node> childNodes

RenderObject extends Node:

* **Mesh (resource)**
  + Vector<float> vertexData
  + Vector<int> indexData
* **Material – NOT a resource. You can compose a shader of these things**
  + Frag shader (string)
  + Vertex shader (string)
  + **VertexLayout**
  + Vector<**Texture (resource)**>
  + Vector<Uniforms (primitive data)>

**GLTF IMPORTER**

Where will this actually output things? It has several outputs – but meshes and materials are probably first. Secondarily, we recurse through the node structure and create node objects. If a node references a mesh, then it is a RenderObject.

Some thoughts:

* Who owns the data? The importer will be creating Mesh objects, but they will go out of scope. Want to avoid a copy
* Problem\* Mesh should have access to the vertex layout as well
* Scenario: what if the GLTF mesh does not contain all the attributes that are required? There needs to be some sort of matching between Mesh vertex layout, and Shader vertex layout. Tricky problem

**RENDER LOOP**

In the renderer, we will have a std::vector<RenderObject>. They need to be sorted so we can do the following loops

for shader in shaders: 🡪 This MUST coincide with the vertex layout

set vertex layout (ONLY if it has changed – likely shaders share the same vertex layout)

set shader

for renderObject in renderObjects:

// this is not really right

// GPUs that support instancing can set the instancing data, single draw call

for instance in instances: // this might only be one, that’s fine

setTransform

submit // this would be multiple drawcalls