Computer Graphics 4052
Lab 4
Binh-Son Hua
Trinity College Dublin

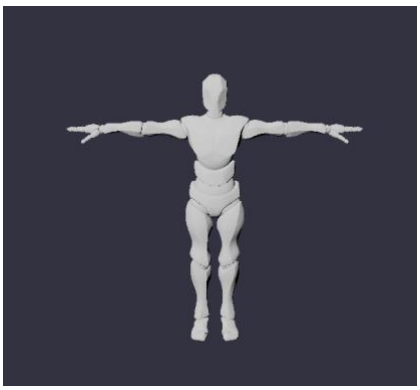
In this lab, we are going to learn how to animate a 3D character in OpenGL. Our objectives are:

- To load a 3D character model into OpenGL using GLTF format
- To animate the character skeleton by applying hierarchical transforms
- To attach the character mesh to the skeleton by linear blend skinning
- To animate the skinned mesh

0. Model loading

In this lab, we will work with 3D data representing a humanoid character. The data includes a static 3D triangle mesh for rendering, the skeleton for animation, as well as additional data for animating the mesh using the provided skeleton. The data is stored in glTF, a standard file format for storing 3D models.

The template code in **lab4_character.cpp** shows you how to load the geometry from a glTF file to OpenGL buffers and perform rendering. By default, you will see a robot (or bot) in a standard pose (the T-pose) as follows.



Based on what you learned in the previous lab, this template code simply uses the tinygltf library to parse a glTF file containing the 3D model, load the geometry into vertex array object (VAO), vertex buffer objects (VBOs), and setup the vertex and fragment shaders to render the model to the screen. A Lambertian illumination model is implemented to illumination this 3D model.

The glTF file follows the JSON file format and stores a list of nodes structured into a hierarchy (a tree data structure) to represent the character skeleton. One of the node is used to specify the mesh data as well as animation data. For a quick reference to the glTF format, refer to this reference card: https://www.khronos.org/files/gltf20-reference-guide.pdf

1. Animation

To animate a 3D model, we will need to bind the character skeleton to the 3D geometry, and then rig the skeleton to certain poses so that the skeleton guides the 3D geometry on how to deform accordingly. This process is usually performed in 3D modelling software, e.g., Blender.

To provide realistic animation, one approach is to capture real motion by human performers and use them to represent the animation of the skeleton. For simplicity, we refer to a public motion capture dataset and retrieve a mocap data file (in BVH format). We load the mocap file and the 3D model to Blender and perform a binding of the skeleton to the 3D model, and export everything to a glTF file. For those who are interested in how to use Blender to produce glTF with motion captured animation, please refer to the appendix.

Our task in OpenGL is therefore to reproduce the animation stored in the glTF file onto our rendering. For simplicity, in this step, let us focus on the animation itself, and set aside the 3D model. We will simply load the animation from the glTF and render to a stick figure.

To animate our model, in the main loop, we have a new update function, which aims to modify the internal states of our 3D models (e.g., deform the model using joints) and then render the 3D model to the screen. The change of the internal states is time dependent so that if we perform update and render repeatedly, we can see our 3D animation!

Our goal here is therefore to consider how to modify the internal states at each time step to produce animation. In our case, as we try to animate a 3D character, the internal states we need to manipulate is the list of joints representing the skeleton.

In this section, use the template code from **lab4_skeleton.cpp**. We represent each joint by a sphere, and connect the joints by lines. At start, you will see all joints at the center with no lines as the transforms of the skeleton joints have not yet been properly constructed.

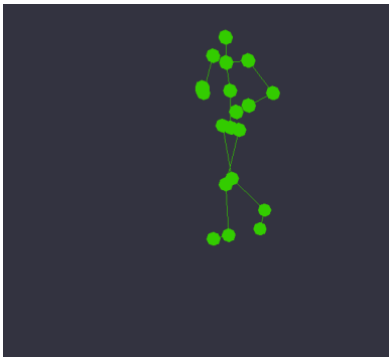
To compute the transforms of the joints, at each time step, we need to compute the 4x4 transformation matrix at each joint. Note that due to the hierarchical structure of the skeleton, a joint has a local transform, and the parent transform by its parent node. If we multiply the parent transform with the local transform, we obtain the global transform of a joint, which transforms the joint to the world space.

During animation, the local transform of a joint will be modified, and the global transforms must be updated to obtain a new skeletal pose.

Based on the template code, your task is to implement the TODOs in **lab4_skeleton.cpp**.

Hint: you can traverse the skeleton by starting from the root joint, which is the node named “Hips”. Note that a node in glTF can store generic information. Not all nodes are used to store skeletal joints.

Once completed, you will see the following stick figure in a running pose.



2. Skinning

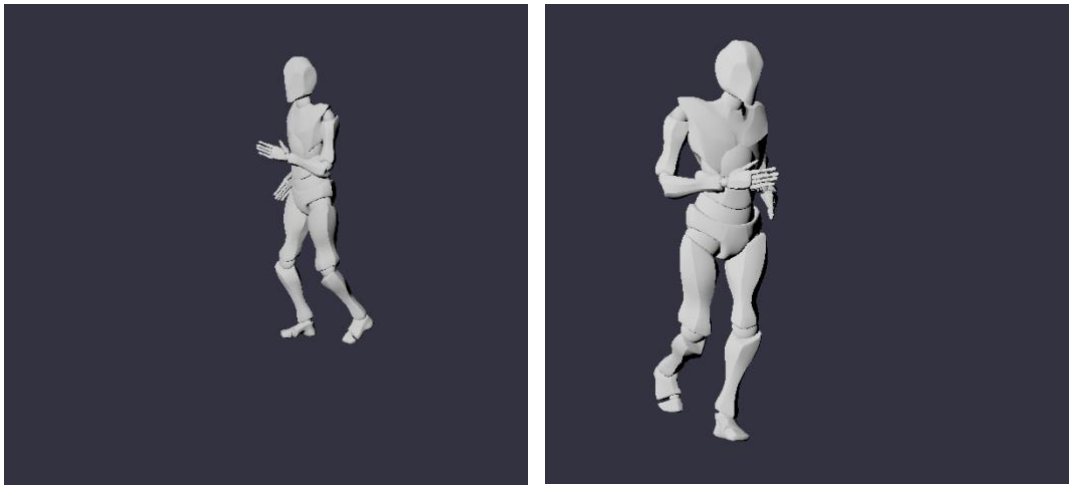
Now we are ready to consider how to drive our 3D character animation using a skeleton. This requires two major steps.

1. Compute a 4x4 matrix for each joint, also known as the joint matrix.
The joint matrix can be established by two 4x4 matrices: the global transform we computed in the previous section, and an inverseBindMatrix that specifies how to transform a mesh vertex into the local space of a joint. The inverseBindMatrix is provided in the glTF file.

2. Given the joint matrices, apply linear blend skinning, which deforms mesh vertices given the global transforms of the joints. In our case, each vertex is controlled by 4 joints.
As the transformation needs to be done on each vertex, we can implement skinning in a shader.

Follow the template code from **lab4_character.cpp**, and implement linear blend skinning.
Hint: Follow the formula in the glTF reference card if needed.

The below screenshots demonstrates the animated 3D character.



Debugging tips

- Skinning and animation can be separately implemented. That means to implement skinning, you can simply disable the update function to keep your character in T-pose. In this case, after adding the skinning matrix to the vertex transformation in the shader, your character will remain in T-pose without any geometric distortion.
- Transformation bugs can be sometimes hard to check, so make sure you set your camera properly (sometimes far enough to observe a wider space is good).
- Check the references for glTF resources and tutorials.

Submission: Package lab4_character.cpp and all shader files, and an mp4 video that captures the rendering of your animated 3D character, into lab4_results.zip. On Blackboard, go to Submissions -> Lab 4 and upload your zip file.

Deadline: **Wednesday, 27 November 2024, at 12pm (noon).**

Marking: You will get a complete/incomplete score based on your results.

A. Creating a glTF model using Blender

Here we list the main steps to create an animated glTF model using Blender. These steps might be useful if you want to create a new animated model with a mocap data of your choice.

1. Import a model to Blender. You can find your preferred 3D model on public asset websites such as Sketchfab, Mixamo, etc.
2. Import mocap data BVH to Blender. BVH is a simple text-based file format for storing motion captured animation data. A BVH file only stores animation data, and it needs to be used in conjunction with a 3D character to represent an animated 3D character. Here we import the BVH file to Blender and then bind the animation to our previously imported 3D model. The BVH animation is displayed by a skeletal model.
3. If you drag the slider in Blender's timeline view, you can see that the 3D skeleton animates, but our 3D character remains in a static pose.
4. Now let's align the skeleton to the model.
   Follow this video: https://www.youtube.com/watch?v=GBSC10euloY
5. We are now ready to bind the skeleton to the 3D model. Under the hood it means we need to compute the skinning weights, which specifies how the 3D vertices should deform according to the controlling skeletal joints. Here we assume that the skinning weights can be computed automatically. Follow this video.
   (Note that in practice, the skinning weights can be imperfect and some visual artists might need to customize weights via weight painting on the mesh vertices).
6. During the animation, if you notice that some mesh vertices are not bound to the skeleton and left behind, you can try selecting these vertices and attach them to some nearest joints, e.g., LeftHand, RightHand. Follow the concept in this video: https://youtu.be/s_29guwY-GI
7. Once done, we now have a skinned mesh. The following additional information is stored on each mesh vertex: the joint indices that influences the current vertex, along with the weight values for each joint. By default, Blender stores 4 joints and 4 weight values for each vertex.
8. Now if you drag the slider in Blender's timeline view, you can see that the 3D character now moves along with the skeleton from the BVH.
9. Export the model to glTF format. Remember to tick the animation box so that animation data is included in the glTF file.

References

1. GLTF quick reference guide: https://www.khronos.org/files/gltf20-reference-guide.pdf
2. A very good tutorial on parsing GLTF file: https://gltf-viewer-tutorial.gitlab.io/initialization/
3. Skinning in GLTF: https://github.com/KhronosGroup/glTF-Tutorials/blob/main/gltfTutorial/gltfTutorial_020_Skins.md
4. Animation in GLTF: https://github.com/KhronosGroup/glTF-Tutorials/blob/main/gltfTutorial/gltfTutorial_006_SimpleAnimation.md
5. A web GLTF viewer for checking: https://gltf-viewer.donmccurdy.com/
6. Public animation data in BVH format:
   Motion capture dataset (SFU, NUS): https://mocap.cs.sfu.ca/
   CMU motion capture dataset (cgspeed):
   https://sites.google.com/a/cgspeed.com/cgspeed/motion-capture?authuser=0
7. Online 3D character model: www.mixamo.com