Lab 10: Skinning

Objective & Goals

- Implement and understand the skinning equation

You may work individually or in groups of 2 - 5. Feel free to work in your GDW groups. Only one person needs to submit, make sure everyone's names are in all of the submitted source files.

The provided fbx file was taken from Mixamo, a website that has several animations and character models. The meshes downloaded from Mixamo may come in several pieces, since our OBJ loader only supports loading a single mesh, you will need to merge the pieces into one mesh.

- https://www.mixamo.com/

There is no starter project this week. Implement this lab in your assignment code or in the work you did last week. There are code snippets provided which may be of help.

Itinerary

Part 1: Programming

This tutorial builds directly on the work from last week. The architecture of the animation is the exact same, the only difference is this time we are implementing SkinnedGameObject::draw to perform the skinning equation. See the lab document from last week if you need a refresher on the data structures used to store the animation data.

The skinning equation is:

$$\vec{v}_i' = \sum_j^N w_{ji} T_j^W (T_j^{Bind})^{-1} \vec{v}_i$$

Deformed Vertex

Sum over j joints of all of this

Weight of joint j on vertex i

Joint to World Transform

Inverse Bind Pose Transform for joint j

Vertex in bind pose

See the lecture slides for a derivation of this equation. The idea is straightforward: transform the vertex to be relative to the influencing joint and apply the joint's transformation (i.e. the HTR animation for the current frame) onto the vertex. You then repeat this process for each joint influencing the vertex, scaling the resulting point by how much influence the joint has on the vertex.

Skin Weights

Skin weights, also known as vertex or joint weights, simply mean how much influence does a particular joint have an a specific vertex. Skin weights can be exported from Maya in XML format. See the "guy_weights.xml" file in the assets/animation folder. The format of the file is:

```
<points> // A list of each vertex in the Mesh
        < pointIndex    pointValueZYX >
        …
</points>

<weights> // A list of weights for each joint
        < pointIndex    jointWeightOnPoint>
        ...
</weights>
```

We do not use the points section because we get our points (vertices) from the OBJ file. The points in the XML file need to correspond to the points exported in the OBJ file. For example, the point with index 0 in the XML file, must be the first vertex listed in the OBJ file. To ensure that the indices line up correctly, we must export our OBJ from Maya.

There is a "weights" section for each joint in the skeleton. This section contains a list of which vertices are influenced by the joint and how much the joint affects the vertex.

The parsing of the XML file is provided for you, see the definition of SkinnedGameObject::initializeSkeletonFromHTR. Go through the logic of this function from top to bottom and make sure you understand what it's doing before continuing. Make sure you understand where the skin weight data is being stored and why.

Skinning

This is how our skinning pipeline will work:

- Invoke the "draw" function on the root game object. The root will then recursively draw each of it's children joints.
- Inside the draw function we will loop through each vertex that the joint influences and apply the skinning equation. We will accumulate and store the resulting vertex in an

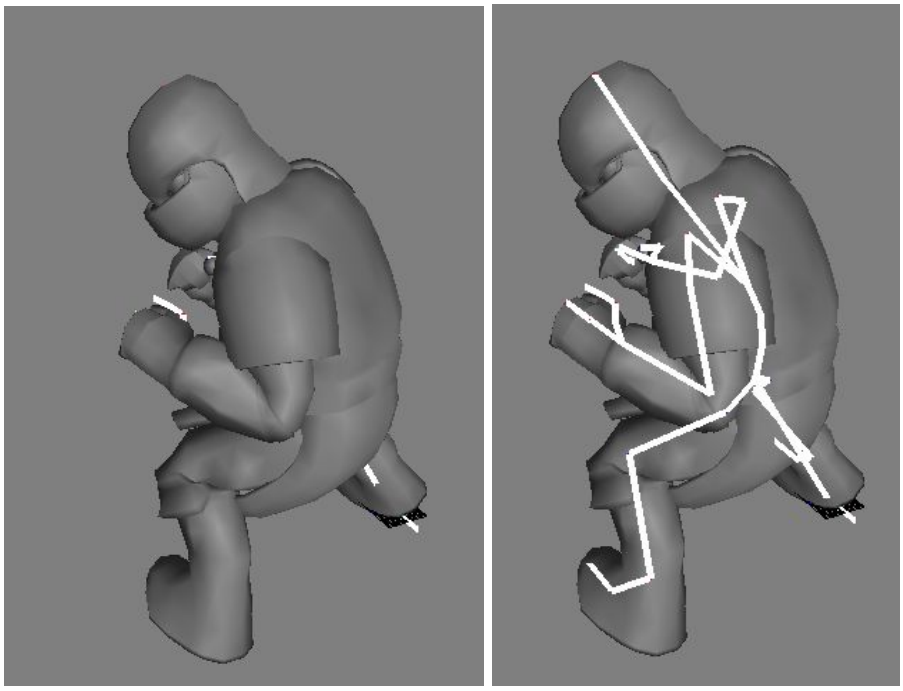
intermediate mesh named “m_pSkinnedMesh”. **We are effectively creating a new OBJ file every frame.**

- After all of the children are updated, we return into the root joint’s stack frame. Once here we invoke OBJMesh::Update. This function is will construct the triangles of the newly generated OBJ mesh. Once the triangles are generated, we can finally draw the mesh!

Your task is to implement the SkinnedGameObject::Draw function to perform skinning on both the vertex and the normals. Remember that normals represent the **orientation** of the vertex, if we apply a rotation on the vertex, we must also rotate the normals to maintain proper lighting. Also remember that normals only represent rotation, so we do not apply the translation or scale of the joint onto the normal. Normals also need to be unit length, so remember to normalize them prior to drawing.

Tip: The above pipeline may seem very confusing, especially the first time you read through it. That’s okay! You will need to work out the logic. I recommend busting out the pencil and paper and deriving what’s happening. Don’t be discouraged if it doesn’t make sense at first, unfamiliar code almost never does. The only way to get better at reading code is by practicing!

The end result should look like this:



Try enabling the texture:

Part 2: Exporting the Data (NOT being marked, just for your knowledge)

The pipeline to create the data needed for skinning is a little complicated:

1. Export an FBX from Blender
2. Load the FBX into Maya
3. Export the OBJ mesh from Maya (make sure mesh is triangulated)
4. Export the skin weights from Maya
5. Load the FBX into 3DS Max
6. Export the HTR animation from 3DS Max

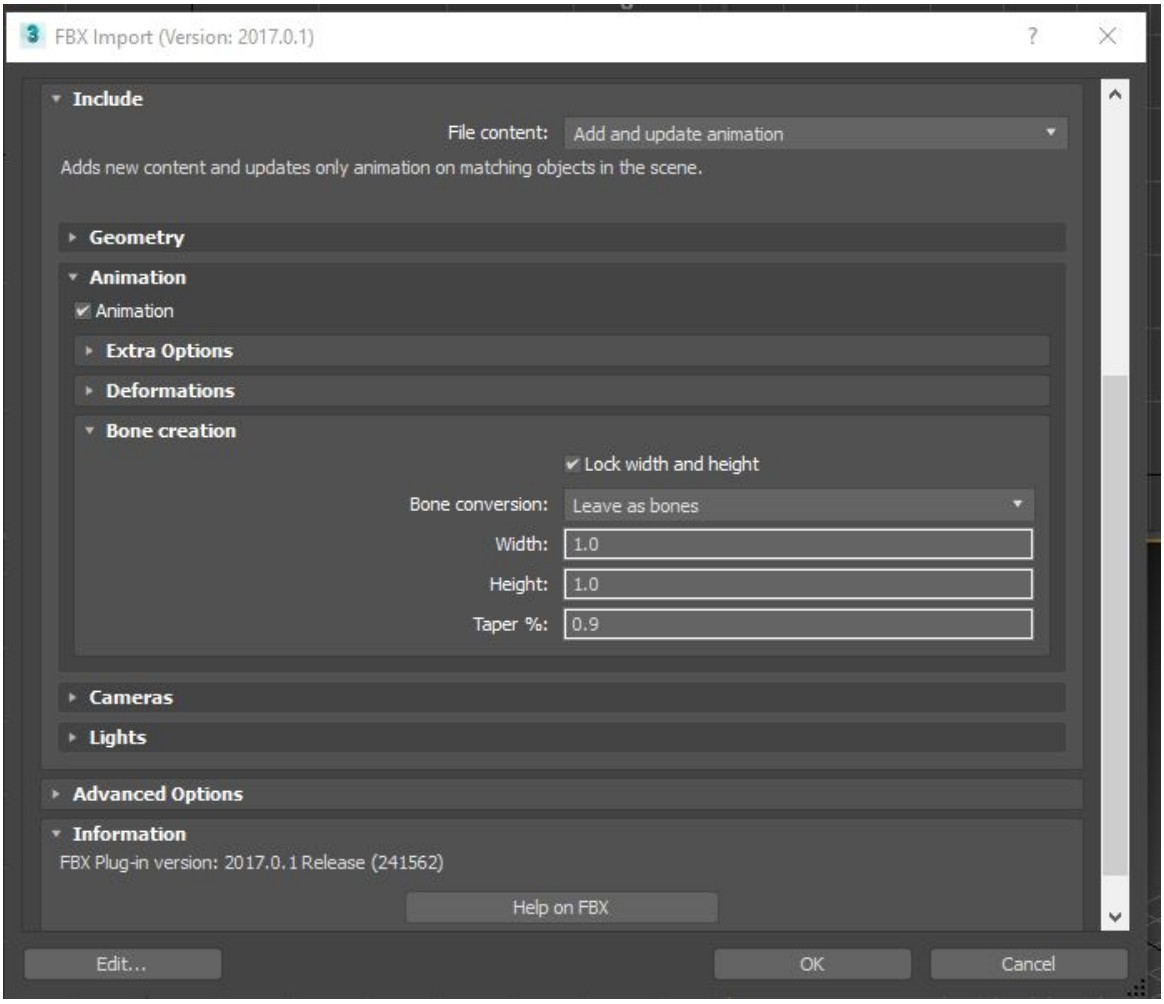
The reason why the OBJ needs to be exported from Maya is because the skin weights must be exported from Maya. The only way to ensure that the indices in the weights file match the indices in the OBJ file is to export both from Maya. For some unknown reason, Maya is the only program that lets you export skin weights directly.

The reason why we need to export the HTR file from 3DS Max rather than MotionBuilder is because MotionBuilder does not export Bind Pose correctly, which is needed for skinning.

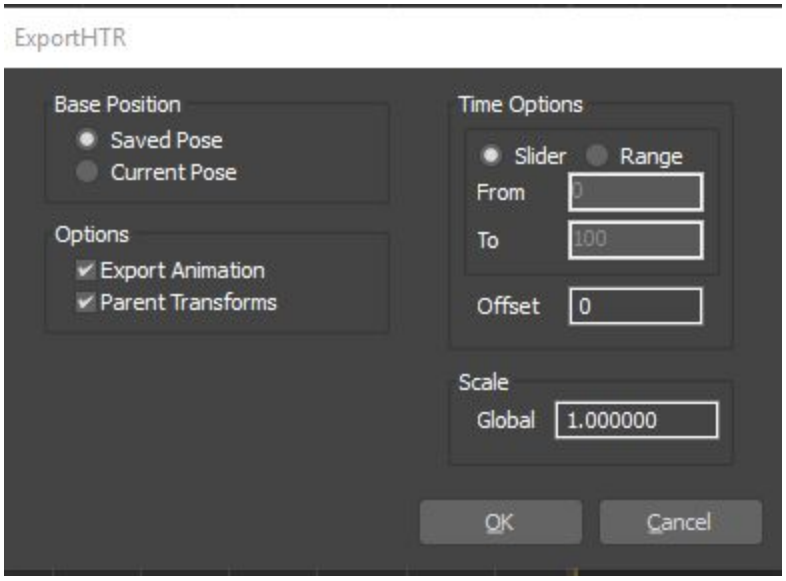
Here are details for the steps:

1. Exporting the FBX is the same process as last week, just select the object, click file > export > FBX. Remember to set the export option to FBX ASCII
2. Open up Maya and click File > Import. Select the FBX exported from blender.
   - Triangulate the mesh if necessary (Mesh > Triangulate)

3. Export the OBJ from Maya: File > Export Selection (left click the character to select it) > in the “File Type” dropdown select OBJexport. In the file options turn everything off except for normals.
   - If the “OBJExport” option is not listed in the dropdown, do the following:
      - Windows > Settings / Preferences / Plug-in Manager. Scroll down until you find “objExport.mll” and enable it by checking the two boxes.
4. Export the weights from Maya: Deform > Export Weights Maps. Enter a filename and hit export.
5. Import the FBX into 3DS Max: click on the “3 Max” button in the top left corner > Import > Select the FBX file. Make sure the import settings for Animation matches the below image:



6. Exporting the HTR file from 3DS Max: click on the “3 Max” button in the top left corner > Export > From the drop down select HTR. The default settings should be fine, hit “okay” on any warning that comes up.



What's Next

- We're done! (This is the last lab)
- If you create a program which combines all of the concepts from these labs, you will end up with a pretty powerful animation editor.
- Holy file formats! We're working with a bunch of different file types, each of which have a specific purpose. You should now be familiar with all of the data in these files (OBJ, HTR, Weights). Going forward you may choose to use a more consolidated format such as FBX. Remember that the data will remain the same, regardless of what file type you're using. So never think "I have no idea what's inside an FBX file, we only worked with OBJ files in class!", it's the same data, just laid out differently! For example, the FBX format takes the data from an OBJ, HTR and SkinWeight file and throws them into one. Sounds great, why didn't we do that from the beginning? Unfortunately parsing an FBX file is not as straightforward as parsing each file individually, in fact it's quite a bit more complicated.

Submission

Submit a zip file the following:

1. Any source files you modify

Make sure you put your name and student number in a comment at the top of each file!
Failure to follow these submission guidelines will result in a **zero**!