Lab 9: HTR Files

Objective & Goals

- Understand how animation data is stored in the **H**ierarchical **T**ranslation-**R**otation (HTR) file format
- Implement an HTR file loader
- Create a hierarchy of GameObjects using the data loaded from the HTR file
- Create a simple joint hierarchy animation in Blender and load it in

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.

Updates to Framework

- TinyXML2 to parse xml in preparation for next week (skin weights are stored in XML)
    - https://github.com/leethomason/tinyxml2

Itinerary

Part 1: Programming

- First, we need to load the animation data contained in the HTR file
- The HTR file format is split into the following sections:
    - [Header]
        - Describes the data in the file, contains things like rotation order, number of animation frames, number of joints, animation frame rate etc.
    - [SegmentNames&Hierarchy]
        - A list of each joint and it's parent
    - [BasePosition]
        - This is bind pose, the transformations in the actual animation frames are expressed relative to this pose
        - Note: when performing skinning, the base mesh should align with this pose. MotionBuilder seems to export this section incorrectly. For next week, install 3DS Max. 3DS Max exports good HTR files.
    - Joint Animations
        - In this section, the animation frames for each joint are listed
        - The data is organized as:
            - FrameNumber t_x t_y t_z r_x r_y r_z s_{xyz}
            - Where 't' is translation, 'r' is rotation (euler) and 's' is a uniform scale value

- A nice thing about the HTR file format is that the sections are guaranteed to be in this order, this simplifies the parsing process.
- You should open the provided HTR file “simple_rig.htr” (located in assets/animations/) and examine it’s contents. Compare the data in this file with the description given above. Simple_rig.htr is a very simple HTR file, take a look at “jog.htr” to see what an actual mo0cap file looks like.

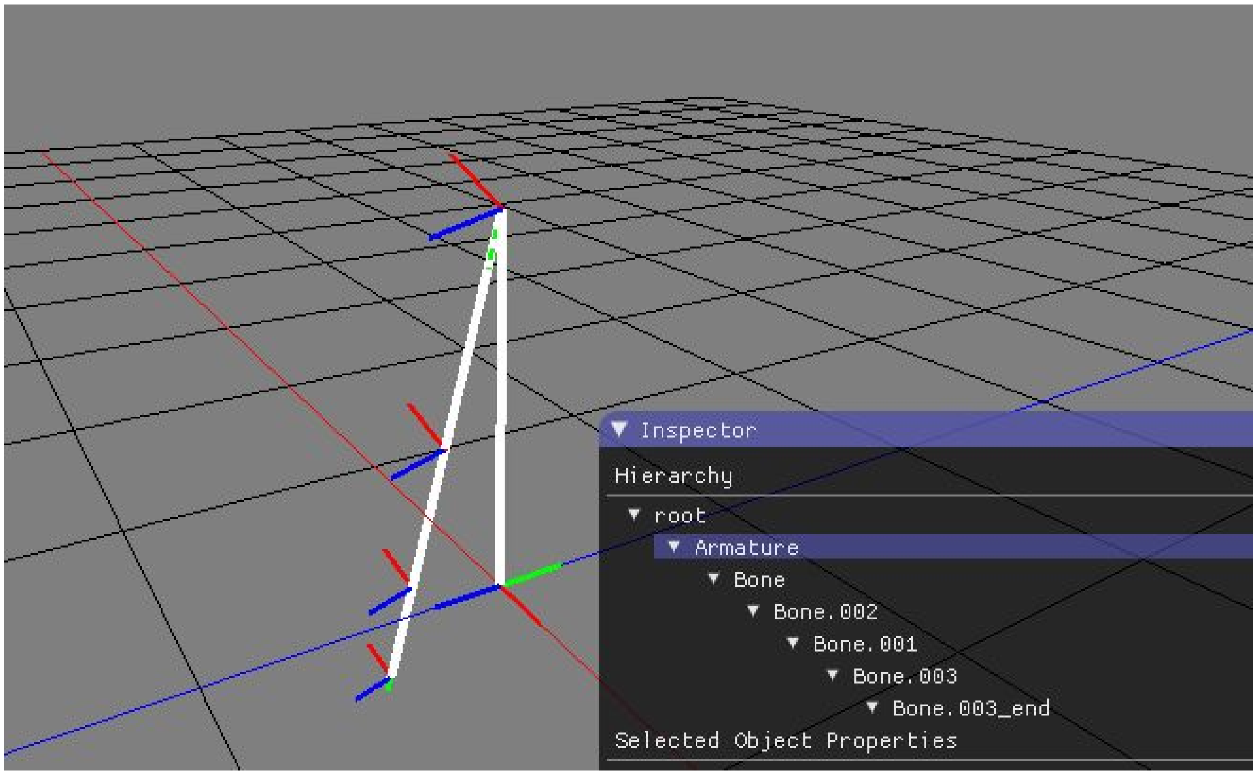
The architecture for the animation system is as follows:

- The HTRLoader class (located in HTRLoader.h) will parse the HTR file and store the data in “JointDescriptor” objects.
- The JointDescriptor struct (located in JointTypes.h) contains member variables which describe a specific joint, including each frame of animation. The HTRLoader class will create a JointDescriptor object for each joint loaded in from file.
- We now have a “SkinnedGameObject” class which inherits from the GameObject base class. This class overwrites the update function defined in the base class to set the translation, rotation and scale (TRS) of the object using the data from the HTR file. For now we are just focusing on the HTR animation, next week we will overwrite the draw function to actually perform the skinning.
- In the InitializeScene function located in main.cpp, we invoke SkinnedGameObject::initializeSkeletonFromHTR, this function will create an instance of the HTRLoader class and invokes HTRLoader::loadHTR which loads the animation data into memory. Once the animation data is in memory, we construct GameObjects for each joint.

Todo list:

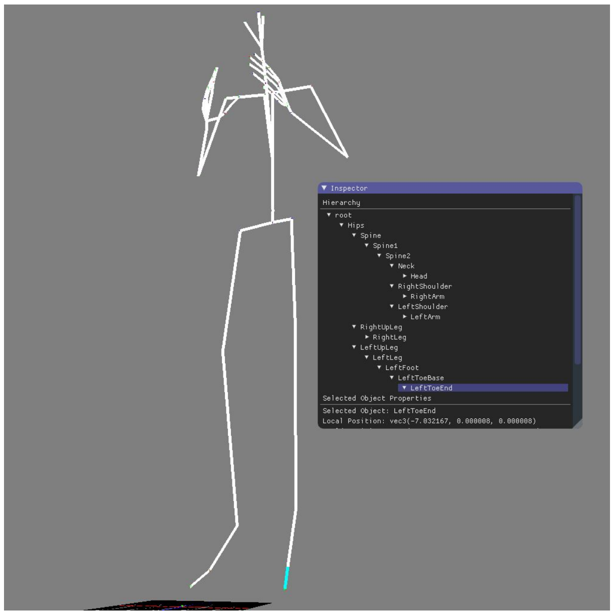
- Open up JointTypes.h, in here are some structs which we will use to store the data loaded in from file. Compare these structs with the HTR file to get an idea of where the data from file will be stored.
- Open up HTRLoader.h, here you will find several functions which will facilitate the parsing of the HTR file. loadHTR is the main function we will be working with. Open up it’s definition and take a look at what’s happening.
- The “Header” and “SegmentNames&Hierarchy” sections of the HTR file are provided for you. Take a look at the “processHeaderSection” and “processSegementNameSection” functions to see how the data is parsed.
- Your first task is to implement the “processBasePositionSection” and “processAnimationSection” functions. You need to parse the data and store it in the appropriate place.
- Now we should have the animation data loaded into our program. Next we need to create actual GameObjects using this data. This is the exact same process we did last week, but this time, instead of creating the GameObjects manually, we create them using the data from the HTR file. Do this in the HTRLoader::createGameObjects function.

- Once you create the GameObjects, you can run the project. It won't work just yet but you should be able to see the hierarchy in the GUI.
- The next step is to create the transformation matrices for each GameObject in SkinnedGameObject::update.
- Now you can run the program, you should see something like this:



```
▼ Inspector
Hierarchy
 ▼ root
   ▼ Armature
     ▼ Bone
       ▼ Bone.002
         ▼ Bone.001
           ▼ Bone.003
             ▼ Bone.003_end
Selected Object Properties
```

  - If you do not see this, you've made an error parsing which you should fix before continuing.
- The hierarchy will not move, that's okay because this HTR file has no animation! ,
- Load in the "jog.htr" file. You should see an animation of a walk cycle which looks like this:

- 

```
▼ Inspector
Hierarchy
▼ root
  ▼ Hips
    ▼ Spine
      ▼ Spine1
        ▼ Spine2
          ▼ Neck
            ► Head
          ▼ RightShoulder
            ► RightArm
          ▼ LeftShoulder
            ► LeftArm
    ▼ RightUpLeg
      ► RightLeg
    ▼ LeftUpLeg
      ▼ LeftLeg
        ▼ LeftFoot
          ▼ LeftToeBase
            ▼ LeftToeEnd
Selected Object Properties
Selected Object: LeftToeEnd
Local Position: vec3(-7.032167, 0.000008, 0.000008)
```

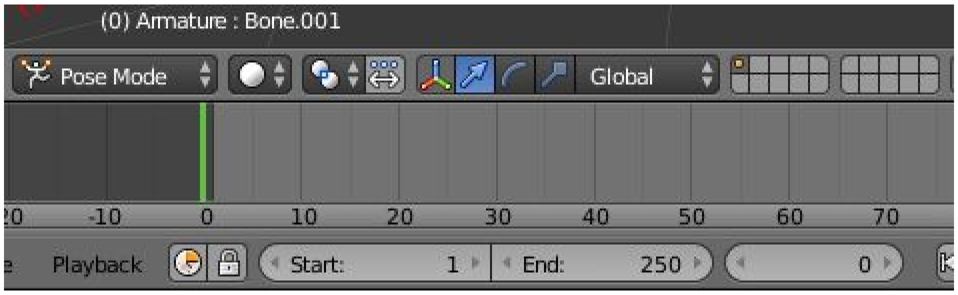
If jog.htr plays and looks correct, then you are done this lab!

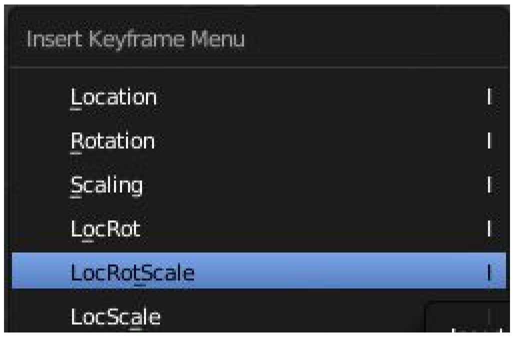
Things to think about:
Notice that the character is rather large and the animation seems to be playing to quickly. How might you fix this? Try playing around with the provided “jog.fbx” file. Also try loading in “run.htr”, notice that this animation has a translation on the root node, this makes it kind of difficult to control the locomotion of this animation, think about how you would fix this.

Part 2: Blender

- Note: This section is not going to be marked as a part of this lab, so don’t stress if you do not get to finish it during the lab time. This is more of a “for your knowledge” thing since, it’s a pretty useful thing to know.

Creating a simple rig in Blender:
- First create an Armature:
- You need to be in Object Mode

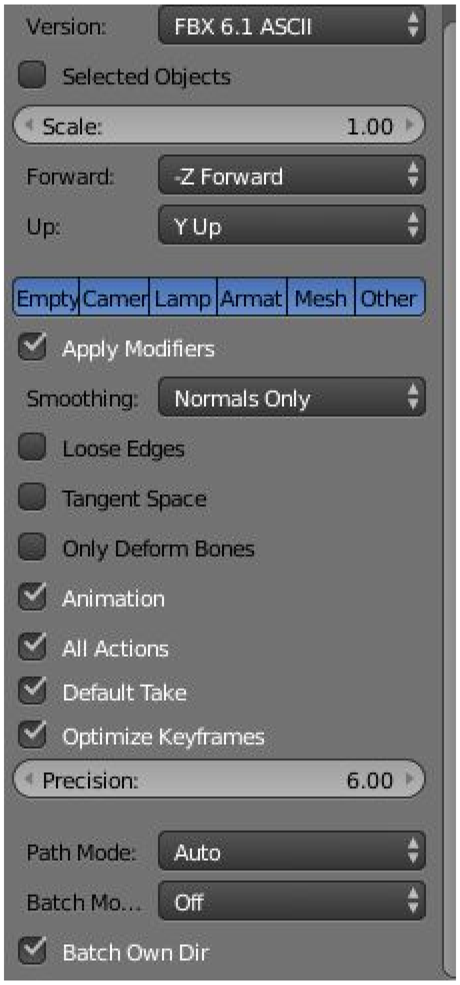
- You can now Pose the armature by going into edit mode and rotating / translating it.
- To create a second joint, select the armature and press the ‘e’ key.
- You can now pose this armature too.
- Once you create a few armatures, you can start creating an animation by going into Pose mode
- Once in pose mode, the first thing you want to do is add a keyframe at the initial position:
- 

- Scrub the the first frame in the animation
- Press the ‘i’ key and click ‘LocRotScale’:
- 

- Notice that the line in the scrubber is now yellow, denoting that a keyframe was placed.
- You can now scrub down to another frame, pose the joint however you like and create a keyframe.

- When you press the ‘play’ arrow, you should see the joint interpolate between the frames.

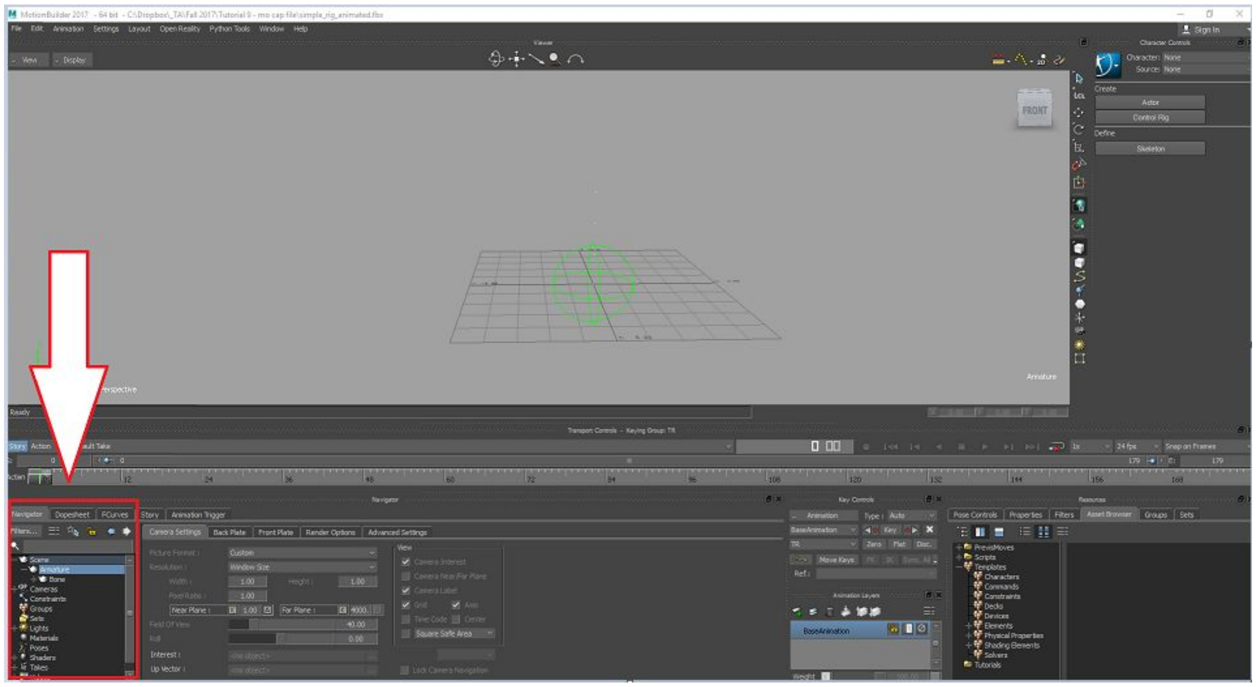
Creating the HTR File

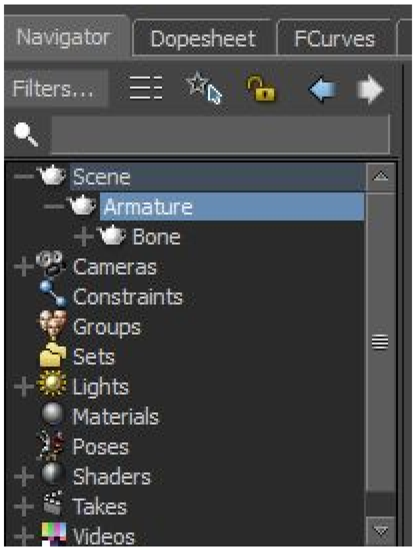
- Once you have a rig you want to export, we need to load it into MotionBuilder to generate the HTR file. To do this, we export the object from blender as an FBX and import it in MotionBuilder.
- In Blender click File > Export > FBX
- In the “Export FBX” options on the left side, make sure your options match these:
- 

Version: FBX 6.1 ASCII
☐ Selected Objects
Scale: 1.00
Forward: -Z Forward
Up: Y Up
[✓ Empty] [✓ Camer] [✓ Lamp] [✓ Armat] [✓ Mesh] [✓ Other]
☑ Apply Modifiers
Smoothing: Normals Only
☐ Loose Edges
☐ Tangent Space
☐ Only Deform Bones
☑ Animation
☑ All Actions
☑ Default Take
☑ Optimize Keyframes
Precision: 6.00
Path Mode: Auto
Batch Mo... Off
☑ Batch Own Dir

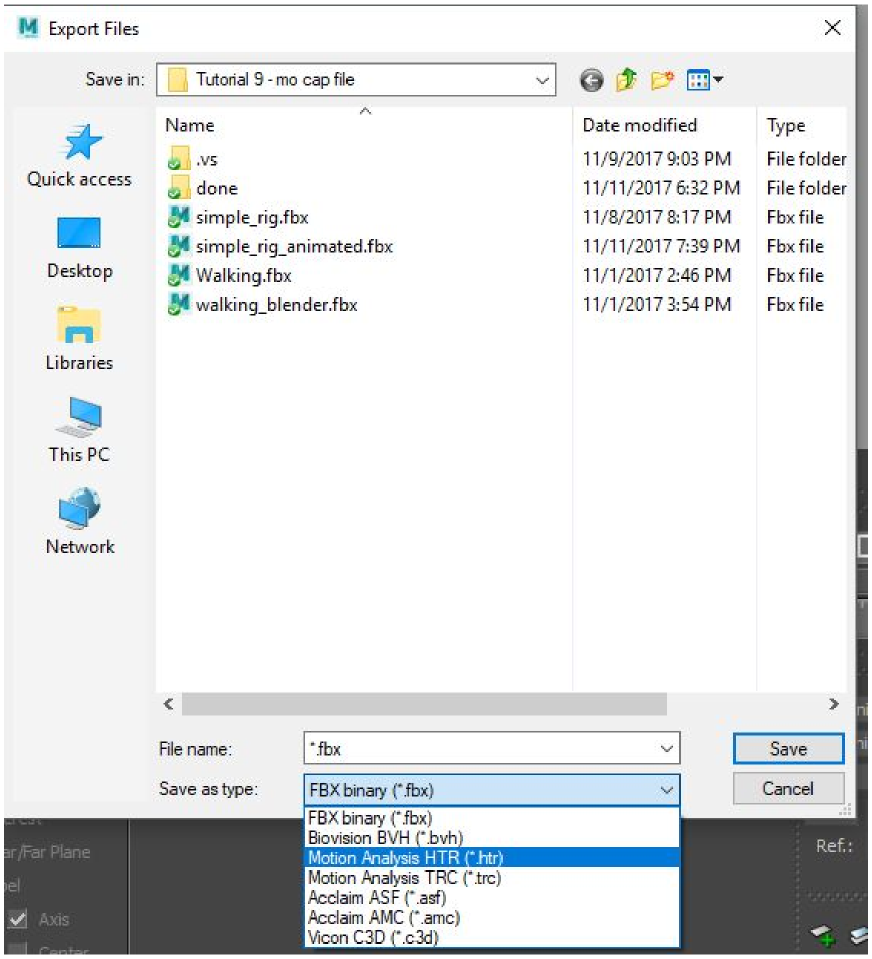
Next you need to open up MotionBuilder and import the file.
Just press “Import” on the dialog box that pops up.

Once the object is imported in MotionBuilder, you need to select the root node of the hierarchy, see the images below:





Once the root node is selected, click File > Motion File Export and select the desired output directory:



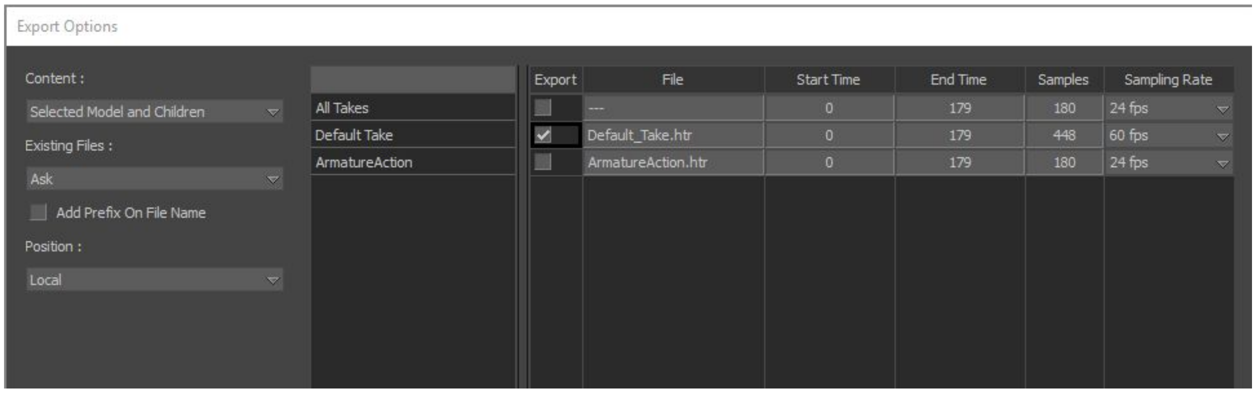
Save in: Tutorial 9 - mo cap file

Name	Date modified	Type
.vs	11/9/2017 9:03 PM	File folder
done	11/11/2017 6:32 PM	File folder
simple_rig.fbx	11/8/2017 8:17 PM	Fbx file
simple_rig_animated.fbx	11/11/2017 7:39 PM	Fbx file
Walking.fbx	11/1/2017 2:46 PM	Fbx file
walking_blender.fbx	11/1/2017 3:54 PM	Fbx file

File name: *.fbx

Save as type: FBX binary (*.fbx)

In the export options you can tell MotionBuilder how you want the data exported. We only need to export the default take. Make sure you set the sampling rate to match your game's FPS.


Export Options

Content : Selected Model and Children

Existing Files : Ask

Add Prefix On File Name

Position : Local

	Export	File	Start Time	End Time	Samples	Sampling Rate
All Takes		---	0	179	180	24 fps
Default Take	✓	Default_Take.htr	0	179	448	60 fps
ArmatureAction		ArmatureAction.htr	0	179	180	24 fps

Note: The "local" option under position doesn't seem to work properly since the "base position" portion of the HTR file is not filled in. For this reason, we will use 3DS Max next week to generate our HTR file. The base position (a.k.a bind pose) is needed for skinning.

What's Next

- You should now be very familiar with how forward kinematics works
- We now know how to create joint animations in Blender and load them into our projects
- Next week we will be looking at deforming meshes based on joint animations (aka skinning)

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**!