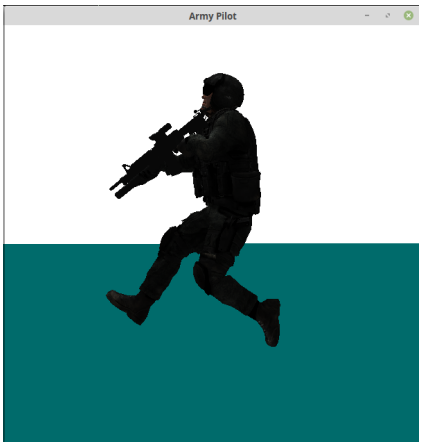
COSC422 Assignment 2

Maree Palmer - 44326398

1. Army Soldier

1.1 Description



The Army Pilot model was the first model I attempted, and so the challenges I solved were the base of the other two sections of this assignment. To transform the vertices of the model to display the correct figure, I got the offset matrix of every bone in each mesh of the model. These offset matrices acted as the product matrix for each bone. Then, the node in the scene with a corresponding name to the bone was found, and the transformation matrix of each node was multiplied by the product matrix, with the product matrix being set to this value. After the final product matrix was calculated, the normal matrix was found by inversing and transposing the product matrix. The transformations of the vertices and normals in each bone were then set by multiplying each vertex and normal by the product and normal matrices respectively.

The model moved as it ran, and so I had to dynamically alter the camera position in the display function. The position of the eye coordinates were calculated from the position of the model's root node, with the constant to center the model in the frame subtracted from this. The position looked at is the origin, or the center of the frame. The camera also rotates around the model, and moves towards and away from the model. This was done by the up and down arrow keys increasing and decreasing a variable called 'look_radius'. Calculations for the angle of the camera around the model were performed using the following equations:

$$position_x = look\_radius \times sin\theta \qquad position_z = look\_radius \times cos\theta$$

The value θ was increased or decreased by the user pressing the left and right arrow keys, and so panned around the model. The model was also rotated 90 degrees into a vertical position for displaying.

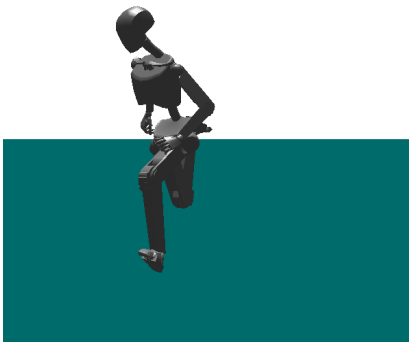
This was the first place I had to alter texture paths, as these pointed to a different folder to where I stored the textures. For simplicity of implementation, I moved the texture files to the same directory as the armyPilot.cpp file. Then, I used the strrchr function to get the section of path.data after the last '/' character. I concatenated this with a '.' character to redirect the target path to the current directory, and set this concatenated string as the path.data object. The textures then loaded correctly.

1.2 Controls

Key	Function
Up arrow key	Move towards the model
Down arrow key	Move away from the model
Left arrow key	Pan left around the model
Right arrow key	Pan right around the model

2. Mannequin

2.1 Description

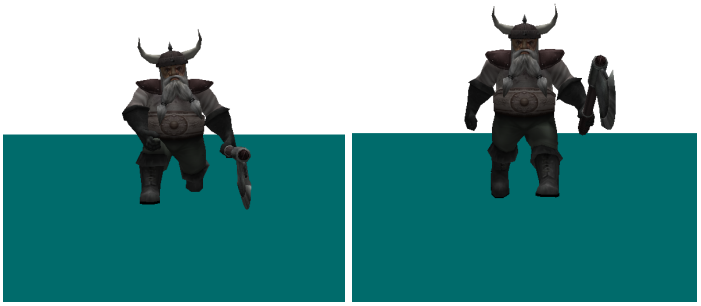


The Mannequin was displayed and transformed similarly to the Army Pilot model. The main difference was that the run animation was loaded from a separate .fbx file. This animation was loaded into a new scene, with the scene duration, the animation transformations, and the model's root position all set from this animation file. The camera was set similarly to the Army Pilot model, where the eye position was set to the centered root node position, with a set radius between the eye and the model. No user-activated movement was present in this animation.

In addition, the animation from the run file had transformations that caused it to run out of frame extremely quickly. This was due to the transformations on the node 'free3dmodel_skeleton'. To prevent this from occurring, transformations of the model from the animation file were only applied if the model name was not 'free3dmodel_skeleton'. After applying this change, the model appeared to move correctly.

3. Dwarf

3.1 Description



The dwarf model has two animations: the animation embedded in the dwarf model file (shown by pressing ‘1’), and a walking animation retargeted from a separate animation file (shown by pressing ‘2’). The base model involved the model crouching down, and looking from side to side, shown in the above left image. This animation was made into smooth, non-jerky movement by interpolating the rotation keys for each step in the animation. The rotation keys did not map one to one with the number of ticks in the scene, and so the movement was very jerky. I found the index of the rotation and position keys that was closest to the current number of ticks, and then retrieved the rotations and number of ticks passed from the current and previous index in the animation’s rotation keys. These values were then used in the Interpolate function of the rotation to make the movement to appear to be between ticks, and the animation was smooth.

To produce the walking animation, I mapped node names in the dwarf model to corresponding channel numbers in the walking animation, as shown below. I mapped the nodes in the legs and torso to give a more natural whole body walking animation. Some of the movements of the original crouching animation are retained.

Node	Channel
head	5
spine1	4
spine2	2
lhip	15
rhip	18
lknee	16
rknee	19
lankle	17
rankle	20

I stored these values in a map NODE_NAMES in the dwarf.cpp file. Then, when the retargeted animation was shown, I checked if the current node was in NODE_NAMES, and if so set the node animation to that from the walking animation at the corresponding channel. Then, I obtained the rotation keys for the animation at the current index (as I would for the crouching animation), and set the rotation to be the value at this index in the animation's rotation keys. In this way, the selected nodes were animated to be walking.

Once I had retargeted the walking animation, I found that the model glitched after one motion of the left and right legs moving had been completed. This was because there were less motions in the walking animation compared to the original crouching animation (less ticks) - 13 compared to 55. To make the walking animation continuous and smooth, I decided to take the index for the rotation keys at tick%13, which caused the animation to smoothly repeat four times in 52 ticks. However, there was a small stutter in movement between the end of these four movements and the end of the crouching animation, as there were three ticks with only a partial amount of walking. I decided to shorten the duration of the animation to 52 ticks to gain the smooth walking animation with minimal stuttering in the crouching animation.

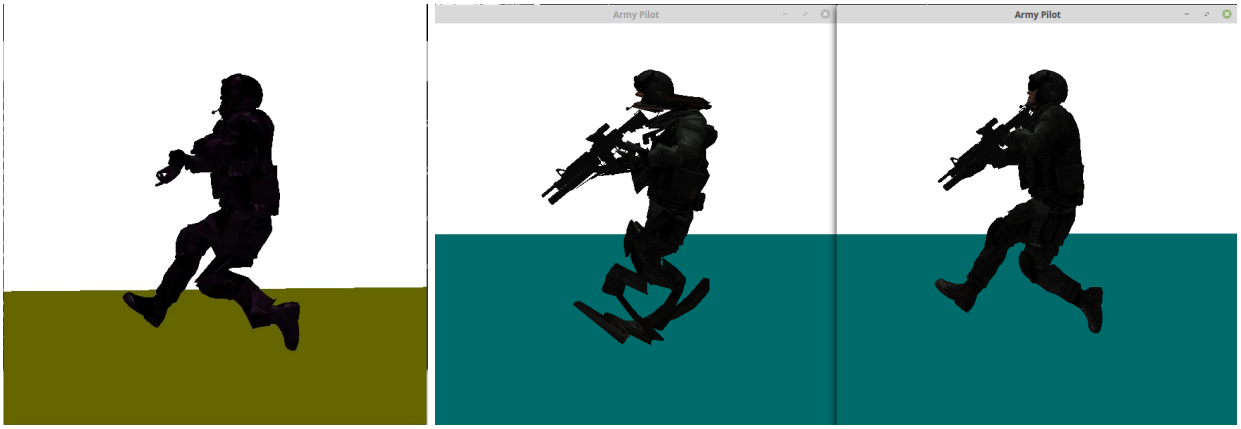
Another challenge I faced was that in the retargeted walking animation the model was translated down from the original animation of crouching. This caused the dwarf to phase through the floor. I discovered this was due to the node titled 'middle', on channel 1. To prevent the model phasing through the floor, when the channel was at 1, I set the index to be at 0, and the model would be translated to its intial position, so no change was observed.

3.2 Controls

Key	Function
1	Play the embedded animation
2	Play the retargeted animation

4. Extra Feature: Vertex Blending

4.1 Description



The extra feature I implemented was vertex blending in the Army Pilot model. As shown in the image on the left, the soldier's limbs do not match as real-life limbs would, and look "broken". This can be solved by taking into account the weights of each bone in the model's meshes at each vertex, rather than using unit weights for each bone.

To perform this, I created an array of aiMatrix4x4 objects to store weighted product matrices for each vertex in each mesh. I then iterated through each bone, and obtained the weight for each bone. I then created an identity matrix for the weight as follows:

```
aiMatrix4x4 weightMat = {
    bone->mWeights[k].mWeight, 0, 0, 0,
    0, bone->mWeights[k].mWeight, 0, 0,
    0, 0, bone->mWeights[k].mWeight, 0,
    0, 0, 0, bone->mWeights[k].mWeight
};
```

This weight was then multiplied by the total product matrix of the bone, obtained as described in section 1.1. This step proved to be vital, as I missed it in the early stages of implementing this feature. The result can be seen in the central image above. The weight was then added to the index of the weight array corresponding to the vertex the bone related to. After all bones were found for the mesh, each vertex in the mesh was multiplied by the total weight matrix at the vertex id's index in the weight array. In this way, the soldier's limbs looked "unbroken", as shown in the rightmost image above.

References

COSC422 Lecture Slides
Assimp class documentation
Provided model, animation, and texture files
COSC422 Labs 11 and 12