Alex Tompkins (ato47)
79004580

COSC422 Assignment 2

Army Pilot

This program renders the *ArmyPilot.x* model with its appropriate textures on a floor of simple rectangles. The model is animated using the motion data embedded within the file. Use the controls listed below to view the model from different angles.

Figure 1: model is displayed with appropriate textures

Figure 2: the embedded animation shown from a different angle

Controls

UP/DOWN: Moves the camera towards/away from the model.
LEFT/RIGHT: Rotates the camera left/right around the model.
SPACE/X: Moves the camera upwards/downwards.

Features and Challenges

Texturing

The model was textured using the provided texture files ‘body01.png’, ‘head01.png’ and ‘m4tex.png’. As absolute paths are hardcoded in the ‘ArmyPilot.x’ file, I needed to change these into relative paths at runtime before the files are loaded.

Animation of embedded motion data

Implementing the animation embedded within the model file required two steps. Firstly, I added a *meshInit* struct to store the initial mesh data loaded from the file. It is necessary to persist this data because the transformation for each frame must be applied to the initial vector for the vertex.

The second important part was the implementation of a function *transformVertices* to transform mesh vertices with the matrices from the data. I followed the pseudo-code from the lecture notes[1] to iterate through every bone of every mesh, first getting the node corresponding to the bone and then traversing back up the scenegraph, pre-multiplying by each parent node's transformation matrix. This allowed me to form a matrix product and normal matrix with which to transform vertices attached to the bone.

Vertex blending using weights

Simply animating the vertices attached to each bone using their unit weights can cause unintended visual artifacts like those shown in figure 3, as the vertices attached to multiple bones have only been transformed by one bone.

Figure 3: without vertex blending

Figure 4: with vertex blending

To resolve this, I implemented a vertex blending method. This works by summing up the weighted transformations (one for each attached bone) for each vertex. The total transformation is then applied at the end.

Translational movement

To simulate the translational movement implied by the model's animation, I changed the model's horizontal position by a set amount each frame. Once the model reaches the edge of the floor plane, his position is reset to the other side.

Adjustable camera angle and model tracking

I implemented simple camera tracking of the model and gave the user the ability to adjust the camera angle. Basically, the camera's position is calculated by adding an offset to the model's position. As shown in figure 5, the x and z offsets can be calculated by the cosine and sine of angle θ multiplied by the radius r. Both θ and r can be adjusted by the user using the arrow keys.

[1] Lec09_CharacterAnim.pdf - pg 20

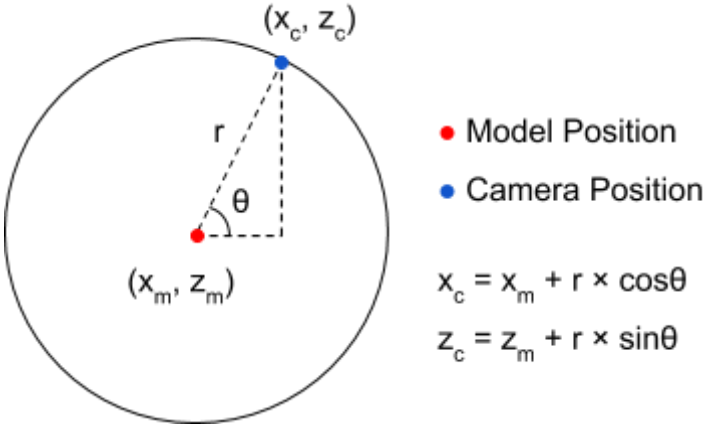

Figure 5: Calculation of the camera's x and z coordinates

Mannequin

This program renders the *mannequin.fbx* model with a single material colour on a floor of simple rectangles. As the model file contains no animation, it is animated using the motion data loaded from the *run.fbx* file. Once again, use the controls listed below to view the model from different angles.

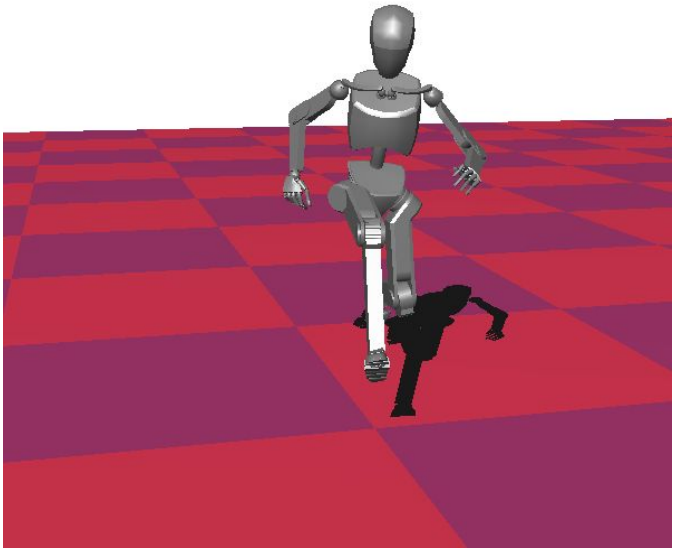

Figure 6: model is displayed with separate animation

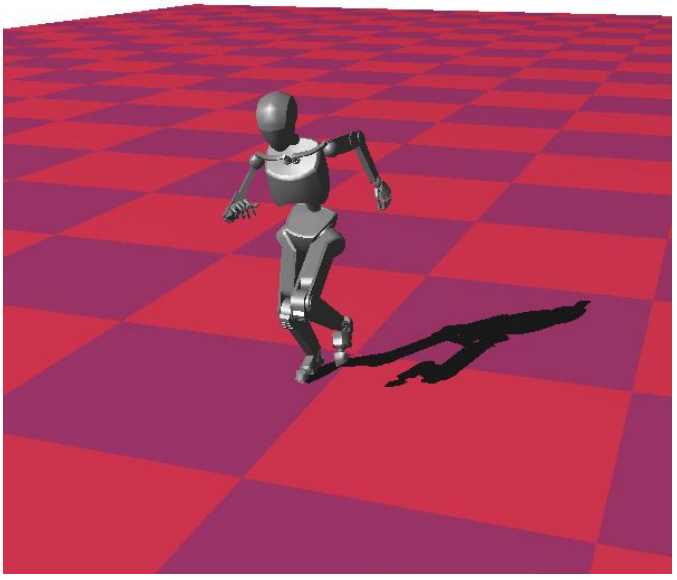

Figure 7: different angle clearly showing planar shadow

Controls

UP/DOWN: Moves the camera towards/away from the model.
LEFT/RIGHT: Rotates the camera left/right around the model.
SPACE/X: Moves the camera upwards/downwards.

Features and Challenges

Animation of a rigged character model using a separate motion file

As the animation data was in a different file, I loaded this as an additional assimp scene. As the node names matched up in both files, adding the animation to the existing model was simply a case of replacing usages of the default scene with the animation scene in some existing code:

- When calculating the animation's duration
- When updating node matrices
- When transforming vertices, so the correct transformation matrix is calculated

Planar shadows

I decided to implement the planar shadows extra feature for this model. This was done by rendering the model an additional time with lighting disabled and a uniform grey colour, using the shadow matrix provided in the notes[2] to project vertices onto the *y=0* plane. The resulting flat surface is translated slightly upwards in the *y* direction to avoid z-fighting artifacts with the floor.

Dwarf

This program renders the *dwarf.x* model with its appropriate textures on a floor of simple rectangles. It is animated using the motion data embedded within the file, and an additional animation from the *avatar_walk.bvh* file has been retargeted for this model. Use the controls listed below to view the model from different angles and toggle between animations.

Figure 8: model displayed with embedded animation

Figure 9: model using separate walk animation

[2] PlanarShadows.pdf - page 2

Controls

UP/DOWN: Moves the camera towards/away from the model.
LEFT/RIGHT: Rotates the camera left/right around the model.
SPACE/X: Moves the camera upwards/downwards.
1: Switch to the animation embedded in *dwarf.x*.
2: Switch to the motion capture animation defined in *avatar_walk.bvh*.

Features and Challenges

Interpolating between non-uniform keyframes

An issue with both animations is that they are defined with non-uniform keyframes. This means that the index of the position/rotation keys does not necessarily correspond to their time (in ticks). Furthermore, there are some times for which no rotation keys exist. Therefore, I first needed to implement a method that chooses keyframes to satisfy this equation:

$$rotationKeys[i-1].mTime < t \leq rotationKeys[i].mTime$$

Once these consecutive rotation keys are found, we can interpolate between them by first determining the interpolation factor:

$$factor = \frac{t - time_{i-1}}{time_i - time_{i-1}}$$

Closer to 0 means more of the earlier keyframe, closer to 1 means more of the later one. The corresponding quaternions for the two keyframes are then interpolated using this factor as described in the lecture notes[3].

Animation retargeting

Only some of the channels in the walking animation were used to animate the dwarf model. This was done by constructing a map of node/channel names in the model to animation channel numbers in the secondary file. I matched these up via trial and error, e.g. experimenting to determine that *lankle* corresponded to *lFoot*, *rhip* to *rThigh*.

I chose to use only the channels for the lower body and torso because the bind position for the two animations differs at the shoulder. This means that if the secondary animation data for the arms is used, the joints bend at unnatural angles. If a node is not mapped to a matching channel in the secondary animation, its original animation data is used. This causes the dwarf to periodically look around as he walks, just as he does in the embedded animation.

The fact that the two animations have differing durations had to be taken into account. This was dealt with by using a number of ticks relative to the animation, calculated as the modulo of the current tick by the duration of that specific animation.

Two final adjustments were needed. Firstly, when the walk animation is active, only the first position key is used. This is so the model doesn't kneel when walking. Secondly, the model's translational motion is enabled only when the walking animation is used.

[3] Lec09_CharacterAnim.pdf - pg 25

Alex Tompkins (ato47)
79004580

Resources

I only used the model file and textures provided in the assignment files. I adapted the *ModelViewer.cpp* code from programming exercises 11 and 12.

Program Setup

To compile the three programs, run the following commands in the root directory of the project:

```
g++ -Wall -o armypilot ArmyPilot.cpp -lm -lGL -lGLU -lglut -lassimp -lIL
g++ -Wall -o mannequin Mannequin.cpp -lm -lGL -lGLU -lglut -lassimp -lIL
g++ -Wall -o dwarf Dwarf.cpp -lm -lGL -lGLU -lglut -lassimp -lIL
```

Then to run the executables, use the following commands:

```
./armypilot
./mannequin
./dwarf
```