CITS3003 Graphics and Animation
Project Report (Part 2)

Ridge Shrubsall (21112211)

Functionality

The model animation and movement features are fully functional and appear to work without any problems. My partner Even and I worked on tasks A-D together, and I implemented circular and bouncy movements for task E and an extra dancing animation for task F as my individual functionality.

Implementation

a) Texture scaling

```
15 uniform float texScale;
(…)
54      fColor = color * texture2D(texture, texCoord * texScale) + vec4((specular1 / len)
+ specular2, 1.0);
```

To implement the texture scaling menu item, the texScale variable was added to the fragment shader (replacing the default scale of 2.0). The starting point code already sets the texScale variable appropriately.

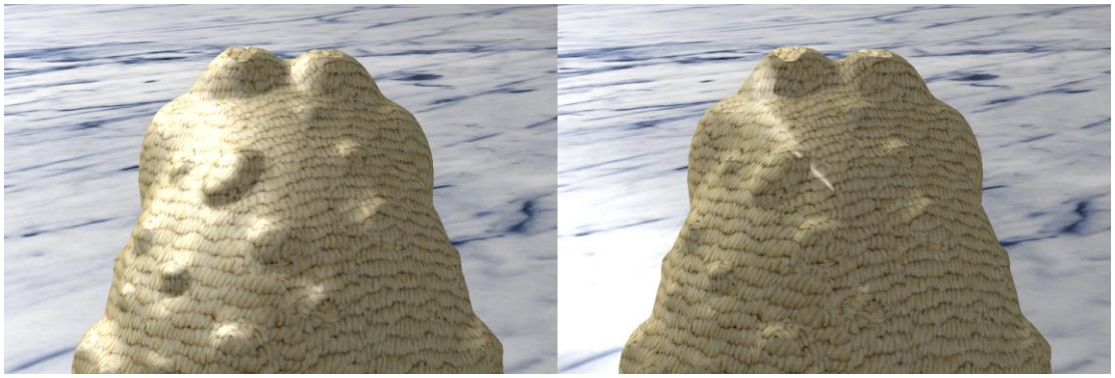
b) Modelling and animation

The gingerbread man model was made in Blender 2.69 by following the provided instructions for the "*Your First Animation*" tutorial. No issues were encountered in creating the mesh or armature rig.

c) Exporting

I went ahead and added the animation support first before getting to this stage to ensure that it worked properly with the test model (the monkey head). After doing an export for the first time, the gingerbread man animation worked perfectly except for the fact that the model ended up halfway through the floor. Whoops! Went back and re-exported the model above the ground plane and it all worked fine.

g) Bug fix for frustum and perspective functions



`c[3][3] = 0.0;` `c[3][3] = 1.0;`

If the last entry (row 4, column 4) in the projection matrix is not initialised to zero, then the lighting breaks when viewing close-up as *w* should be zero for the lighting vector coordinates. The camera view is unaffected by making this change.

d) Adding animation support

```
355     loadTextureIfNotAlreadyLoaded(sceneObj.texId);
356     loadMeshIfNotAlreadyLoaded(sceneObj.meshId);
357
358     aiMesh *mesh = meshes[sceneObj.meshId];
359     const aiScene *scene = scenes[sceneObj.meshId];
360
361     float poseTime = 0.0f;
362     float walkTime = 0.0f;
363     if (sceneObj.meshId >= 56) {
364         double animCycles = 3.0;
365         double elapsedTime = glutGet(GLUT_ELAPSED_TIME) / 1000.0;
366         double animDuration = getAnimDuration(mesh, scene, 0);
367         double animTime = fmod(elapsedTime * sceneObj.walkSpeed, 2 * animCycles *
animDuration);
368         poseTime = fmod(animTime, animDuration);
369         if (animTime >= animCycles * animDuration) {
370             poseTime = animDuration - poseTime;
371         }
372         walkTime = animTime / (animCycles * animDuration);
373         if (walkTime >= 1.0) {
374             walkTime = 2.0 - walkTime;
375         }
376     }
(...)
392     // [B] Set the model matrix.
393     mat4 rot = RotateX(sceneObj.angles[0]) * RotateY(sceneObj.angles[1]) *
RotateZ(sceneObj.angles[2]);
394     vec4 s = rot * vec4(0.0, 0.0, walkTime * sceneObj.walkDist, 0.0);
395     mat4 model = Translate(sceneObj.loc + s) * rot * Scale(sceneObj.scale);
```

The animation time of an object is calculated based on the number of seconds elapsed and its current speed, and is constrained to lie between zero and six times the duration (i.e. for the animation to make three steps forward and three steps backward, we need it to loop six times). The pose time then goes from zero to the duration repeatedly for the first half of the animation and from the duration back to zero repeatedly for the second half. The walking time also ranges from zero to one for the first half of the animation and from one back to zero for the second half.

The object is then displaced by its walking distance multiplied by the walking time. The pose time is used later on in the drawMesh function to calculate the bone transforms.

```
596 static void adjustWalkSpeedDist(vec2 walk_sd) {
597     SceneObject *obj = &sceneObjs[currObject];
598     obj->walkSpeed = max(0.0f, obj->walkSpeed + walk_sd[0]);
599     obj->walkDist = max(0.0f, obj->walkDist + walk_sd[1]);
600 }
(...)
613     } else if (id == 60 && currObject >= 0) {
614         setToolCallbacks(adjustWalkSpeedDist, mat2(24.0, 0.0, 0.0, 5.0),
615                 adjustWalkSpeedDist, mat2(24.0, 0.0, 0.0, 5.0));
(...)
709     glutAddMenuEntry("Walk Duration/Distance", 60);
```

A new menu item to adjust the walking speed and distance was also added.

Note: The speed adjuster uses an increment size of 24.0, as it is the number of frames per second for the exported animation.

Individual implementation

e) Motion type

```
365     bool circle = (sceneObj.motionType == 1);
(…)
370         double animTime = fmod(elapsedTime * sceneObj.walkSpeed, (circle ? 1 : 2) *
animCycles * animDuration);
```

When walking in a circle, the animation only needs to run forwards.

```
395     // [B] Set the model matrix.
396     mat4 rot = RotateX(sceneObj.angles[0]) * RotateY(sceneObj.angles[1]) *
RotateZ(sceneObj.angles[2]);
397     vec4 s;
398     if (sceneObj.motionType == 1) {
399         // Circular
400         float r = sceneObj.walkDist / 2;
401         s = rot * vec4(cos(2 * M_PI * walkTime) * r, 0.0, sin(2 * M_PI * walkTime) * r,
0.0);
402         rot *= RotateY(360 * -walkTime);
403     } else if (sceneObj.motionType == 2) {
404         // Bouncing
405         s = rot * vec4(0.0, abs(sin(3 * M_PI * walkTime)) * 0.3, walkTime *
sceneObj.walkDist, 0.0);
406     } else {
407         // Straight line
408         s = rot * vec4(0.0, 0.0, walkTime * sceneObj.walkDist, 0.0);
409     }
410     mat4 model = Translate(sceneObj.loc + s) * rot * Scale(sceneObj.scale);
```

For the circular walking animation, the object is displaced by $r \cos(2\pi t)$ on the X axis and by $r \sin(2\pi t)$ on the Z axis, and also turned to face the direction that it is walking in.

For the bouncing animation, the object is displaced by $0.3 \times \text{abs}(\sin(3\pi t))$ on the Y axis.

```
631     } else if (id == 61 && currObject >= 0) {
632         sceneObjs[currObject].motionType = (sceneObjs[currObject].motionType + 1) % 3;
(…)
727     glutAddMenuEntry("Motion type", 61);
```

A new menu item to change the motion type was also added.

f) “Thriller” dance

In the spirit of Halloween, I added an extra dancing animation for the gingerbread man. This required adding two extra bones for the hands, and then redoing the posing and exporting steps in Blender.

Reflection

After having completed the first project, this one seemed much easier to follow and implement in the time given (mainly due to there being less coding this time around and more designing and thinking). Having a good understanding of the existing code also made adding the animation aspects of the program more intuitive.